Software Design Redocumentation Menggunakan *Reverse Engineering* Untuk Perangkat Lunak

Tugas Akhir

diajukan untuk memenuhi salah satu syarat

memperoleh gelar sarjana

dari Program Studi Informatika

Fakultas Informatika

Universitas Telkom

1301154278

Muhammad Marchell

Program Studi Sarjana Informatika

Fakultas Informatika

Universitas Telkom

Bandung

2019

Software Design Redocumentation Menggunakan *Reverse Engineering* Untuk Perangkat Lunak

Muhammad Marchell[1], Sri Widowati[2], Jati Hiliamsyah Husen[3]

[1,2,3]Fakultas Informatika, Universitas Telkom, Bandung
[1]marchellehcram@student.telkomuniversity.ac.id, [2]sriwidowati@telkomuniversity.ac.id,
[3]jatihusen@telkomuniversity.ac.id

Abstrak

Software documentation **merupakan salah satu bagian terpenting dalam proses rekayasa perangkat lunak. Dokumen ini berperan penting dalam rekayasa perangkat lunak karena dokumen ini menjadi salah satu faktor keberhasilan perangkat lunak yang dibangun. Dokumentasi yang buruk dapat menyebabkan adanya hambatan saat pembangunan perangkat lunak seperti kesalahan pada pembangunan perangkat lunak serta mengurangi efisiensi pada semua tahap pembangunan dan penggunaan perangkat lunak. Salah satu cara untuk mengatasi permasalahan ini adalah dengan melakukan *software redocumentation* dengan menggunakan metode *reverse engineering*. *Software documentation* adalah suatu kegiatan pembuatan dokumen yang digunakan pada lingkup pembangunan perangkat lunak untuk menyampaikan fungsi-fungsi, operasi dan kegiatan kepada *stakeholder*. Sedangkan *Reverse engineering* adalah sebuah proses menganalisa sebuah sistem untuk mengidentifikasi komponen dan keterkaitan sistem tersebut, dan membuat representasi sistem tersebut dalam wujud lain. Pada penelitian Tugas Akhir ini telah dilakukan proses *software design redocumentation* pada studi kasus aplikasi yang bernama Sisca menggunakan metode *reverse engineering* yang akan menghasilkan dokumentasi desain, serta menganalisa masalah yang terdapat pada implementasi metode dan akurasi data yang dihasilkan dari metode tersebut. Hasil dari penelitian ini adalah metode *reverse engineering* berhasil menghasilkan data yang akurat dan dapat mengefektifkan waktu modifikasi perangkat lunak, dengan adanya masalah terkait dengan pembuatan diagram yang dipengaruhi oleh pemahaman pengguna metode.**

Kata kunci : ***Reverse Engineering, Software Design, Software Redocumentation, Unified Modelling Language,*** **Perangkat Lunak**

Abstract

Software documentation is one of the most noteworthy aspects of software engineering. This document has an important role in software engineering because it is one of the factors in the success rate of developing a software. Poor documentation will result some obstacles in software development process, like development errors and reducing efficiency on every software development process and use. One of the ways to resolve this problem is by doing software redocumentation process with reverse engineering method. Software documentation is a document-making activity that used in software development environtment to deliver functions, operations and activities to the stakeholder. While reverse engineering is a process to analyze a system in order to indentify components and linkages in that system, and make a representation of the system in other form. In this study, we have done software design redocumentation process with a case study of an app called Sisca using reverse engineering method that generated design documentation, and analyze problems that arise and data accuracy that was generated by the process. The results of this research are the reverse engineering method succeeded in producing accurate data and can streamline the time of software modification, with the problems associated with making diagrams that are influenced by the understanding of the method users.

Keywords: Reverse Engineering, Software Design, Software Redocumentation, Unified Modelling Language, Software

1. Pendahuluan

Latar Belakang

Pada masa kini, aplikasi perangkat lunak sudah menjadi bagian penting dalam kehidupan masyarakat modern, baik itu di sektor pribadi maupun di sektor lingkup pekerjaan [1]. Hal ini menyebabkan terus meningkatnya pembangunan dan pengembangan aplikasi perangkat lunak pada tiap harinya. Menurut Philip Laplante, rekayasa perangkat lunak atau yang biasa disebut *software engineering* adalah salah satu bidang profesi yang mendalami secara sistematika cara-cara pengembangan perangkat lunak termasuk pembuatan, pemeliharaan, manajemen organisasi pengembangan perangkat lunak dan manajemen kualitas [2]. Bidang ini merupakan salah satu bidang yang didalami dalam dunia informatika.

Salah satu hal penting yang menjadi bagian dalam *software engineering* adalah *software documentation*. *Software documentation* adalah suatu kegiatan pembuatan dokumen yang digunakan pada lingkup pembangunan perangkat lunak untuk menyampaikan fungsi-fungsi, operasi dan kegiatan kepada *stakeholder* [3]. Dokumen ini berperan penting dalam rekayasa perangkat lunak karena dokumen ini menjadi salah satu faktor keberhasilan perangkat lunak yang dibangun [3] dan dokumen ini menjadi sebuah alat untuk perencanaan dan pengambilan keputusan pada pembangunan perangkat lunak [3]. Permasalahan yang biasanya timbul atas dokumentasi yang buruk bahkan ketidakadaan dokumentasi perangkat lunak dapat menyebabkan banyak kesalahan pada pembangunan perangkat lunak serta mengurangi efisiensi pada semua tahap pembangunan dan penggunaan perangkat lunak [4]. Sehingga pembangunan perangkat lunak menjadi kurang bahkan tidak sempurna.

Permasalahan tersebut dapat diselesaikan dengan melakukan proses pembuatan dokumentasi ulang dan penyempurnaan dokumentasi tersebut, atau yang biasa disebut *software redocumentation*. Salah satu metode yang mendukung proses ini adalah sebuah metode yang disebut *reverse engineering*. *Reverse engineering* adalah sebuah proses menganalisa sebuah sistem untuk mengidentifikasi komponen dan keterkaitan sistem tersebut, dan membuat representasi sistem tersebut dalam wujud lain atau dengan abstraksi yang lebih tinggi [5]. Proses ini dilakukan dengan cara membedah perangkat lunak yang sudah ada, salah satunya dengan cara menganalisa *source code* dari perangkat lunak tersebut [5]. Hasil keluaran analisa dari proses ini berupa *requirement* dan desain dari perangkat lunak itu sendiri. Data yang dihasilkan dari hasil analisa tersebut menjadi dasar pembuatan dokumen perangkat lunak. Dokumen yang dihasilkan tersebut menjadi dasar untuk pembuatan dan modifikasi perangkat lunak di masa depan.

Topik dan Batasannya

Dalam penelitian ini penulis bermaksud menjawab rumusan masalah yang ada. Rumusan masalah yang muncul dalam penelitian ini adalah:

1. Apa permasalahan yang timbul pada implementasi metode *reverse engineering*?
2. Bagaimana tingkat akurasi data yang dihasilkan untuk pembuatan dokumentasi perancangan perangkat lunak dengan menggunakan metode reverse engineering?

Hasil dari proses implementasi metode *reverse engineering* yang penulis lakukan adalah dokementasi desain dari perangkat lunak yang dianalisis. Sedangkan kasus yang penulis ambil untuk implementasi metode ini adalah aplikasi yang bernama Sisca yang berbasis sistem operasi *android*. Aplikasi ini belum memiliki dokumentasi apapun baik dokumentasi desain maupun *requirement* sebelum penelitian ini dilaksanakan.

Tujuan

Tujuan yang ingin dicapai dalam Tugas Akhir (TA) ini adalah membuktikan tingkat efisiensi proses dan akurasi data yang dihasilkan dari metode reverse engineering dengan studi kasus yang ada.

Organisasi Tulisan

Setelah pendahuluan, terdapat bab 2 studi terkait dimana bab ini berisikan literatur yang mendukung dan berkaitan dengan penelitian ini. Teori yang menunjang penelitian ini pun berada di dalam bab 2. Bab berikutnya yaitu bab 3 berisikan implementasi metode yang digunakan untuk dianalisa. Selanjutnya bab 4 memuat analisis hasil dari implementasi metode yang telah dilakukan. Bab 5 merupakan bab yang memuat kesimpulan dari semua yang telah dilakukan pada penelitian ini.

2. Studi Terkait

Pada penelitian yang dilakukan oleh Eric J. Bryne [5], dilakukan studi kasus penggunaan metode reverse engineering untuk menghasilkan dokumen desain sebuah perangkat lunak. Hal ini bertujuan untuk membantu mempermudah implementasi sebuah perangkat lunak berbasis bahasa pemrograman Fortran ke bahasa lainnya, dalam kasus ini adalah bahasa pemrograman Ada. Proses utama yang dilakukan adalah menghasilkan informasi desain yang dihasilkan oleh ekstraksi source code perangkat lunak dengan bahasa Fortran dan diubah menjadi dokumen desain perangkat lunak tersebut. Kesimpulan yang diambil dari penelitian ini adalah beberapa masalah yang terdapat pada saat melakukan proses reverse engineering, seperti pemilahan perbedaan informasi yang didapat dan kapan sebuah perangkat lunak membutuhkan proses reverse engineering.

Selanjutnya, pada penelitian yang dilakukan oleh mahasiswa St. Francis Institute of Technology [6], dilakukan implementasi metode reverse engineering untuk mendapatkan pola desain pada aplikasi web. Tujuan dari penelitian ini adalah pembuatan sistem pendeteksi pola desain suatu aplikasi yang dapat memudahkan dan membuat proses pengertian kode aplikasi serta perawatan aplikasi menjadi lebih efisien. Implementasi metode reverse engineering digunakan dengan cara pembuatan sistem otomatisasi yang berbasis pada metode reverse engineering. Sistem ini mendeteksi pola desain dengan menggunakan struktur pohon. Secara singkat, sistem diberi masukan kode aplikasi yang ada lalu sistem merubah pola kode tersebut menjadi struktur pohon. Struktur

pohon ini mengandung semua kelas yang ada di kode dengan definisi kelas dari kode masukan. Struktur pohon yang dibuat oleh sistem menjadi dasar dari pola desain aplikasi yang dianalisis. Kesimpulan dari penelitian ini adalah sistem yang dibuat dapat mempermudah pembangun aplikasi dalam mendapatkan pola desain aplikasi, sehingga proses pengertian dan perawatan aplikasi menjadi lebih singkat dan mudah.

2.1 Reverse Engineering

Reverse engineering adalah suatu proses menganalisa sistem untuk mengidentifikasi kompenen dan keterkaitan sistem tersebut, dan untuk membuat representasi sistem tersebut dalam wujud yang lain atau dengan abstraksi yang lebih tinggi [5].

Tujuan proses ini adalah untuk mendapatkan desain suatu perangkat lunak yang dapat dijadikan sebagai dasar pembuatan dokumentasi desain sebuah perangkat lunak. Desain tersebut dapat diperoleh kembali dengan menggabungkan bagian-bagian informasi dari source code perangkat lunak tersebut [5]. Beberapa tujuan dari proses reverse engineering adalah sebagai berikut:

1. *Updating/Correcting*. Dalam jangka waktu tertentu, perangkat lunak butuh untuk diperbarui atau dikoreksi untuk memenuhi kebutuhan dalam waktu berlanjut. Seperti penambahan fitur perangkat lunak, revisi antarmuka, dll.
2. *Lost Documentation*. Hal ini diakibatkan oleh hilangnya sebagian data bahkan seluruh dokumentasi dari sebuah perangkat lunak.
3. *Source Code Analysis.* Hal ini dilakukan untuk menganalisa *source code* dari perangkat lunak dengan tujuan untuk mengetahui bagaimana sistem kerja *source code* tersebut.

Secara lebih rinci, ada 7 tahap dalam proses reverse engineering [5]. Berikut adalah tahap-tahapnya:

1. Pengumpulan Informasi.

Langkah pertama yang harus dilakukan adalah mengumpulkan informasi yang berkaitan dengan perangkat lunak dan sistem secara umum. Sumber informasi dapat berupa source code perangkat lunak, dokumen desain dan dokumen yang berisi tentang simulasi arsitektur sistem secara umum.

2. Memeriksa Sumber Informasi.

Langkah kedua adalah mempelajari bagaimana struktur dan kinerja perangkat lunak lebih rinci. Langkah ini dilakukan untuk memungkinkan perencanaan proses pembedahan perangkat lunak dan meneliti informasi lingkup pengembangan perangkat lunak.

3. Mengekstraksi Struktur.

Langkah ketiga adalah memulai memperoleh kembali desain rinci yang diperoleh dari source code perangkat lunak. Langkah ini terdiri dari 2 tahap yaitu mengekstraksi hierarki panggilan dan menyatakannya sebagai grafik struktur, dan merekam informasi tentang pertukaran data antar node di grafik tersebut. Informasi yang diperoleh direkam menggunakan editor grafik struktur dan editor struktur data pada lingkup pengembangan perangkat lunak.

4. Merekam Fungsionalitas.

Langkah keempat adalah untuk merekam proses fungsionalitas yang terhubung dengan tiap node.

5. Merekam Aliran Data.

Langkah kelima adalah membuat model aliran data. Desain terstruktur memberikan pedoman untuk transformasi diagram aliran data ke grafik terstruktur. Secara intinya adalah untuk mengidentifikasi transformasi data pada desain terperinci dan menggunakan hasil transformasi ini untuk membuat model aliran data.

6. Peninjauan Ulang Desain.

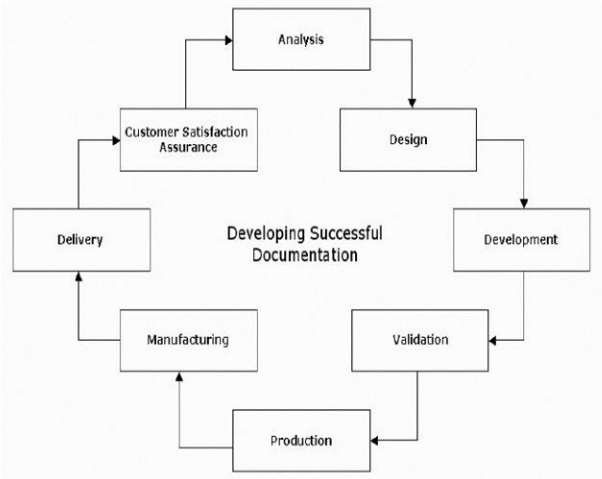
Pada tahap ini, desain yang diperoleh kembali ditinjau ulang untuk tingkat kelengkapannya dan akurasi. Peninjauan ulang untuk tingkat akurasi maksudnya adalah memverifikasi desain yang diperoleh menjelaskan perangkat lunak secara lengkap, tidak ada fungsionalitas yang dihilangkan dan tidak ada informasi yang dilebih-lebihkan.

7. Pembuatan Dokumen.

Langkah terakhir adalah mengolah informasi yang didapat dari hasil analisa dan membuat dokumen berdasarkan semua informasi yang didapat. Informasi yang tidak didapat pada langkah-langkah awal dikumpulkan pada tahap ini. Ini termasuk deskripsi perangkat lunak secara garis besar, fungsi utama perangkat lunak, dll.

2.2 Software Documentation

Software documentation adalah pembuatan dokumen perangkat lunak yang bertujuan untuk berkomunikasi tentang informasi yang berkaitan dengan perangkat lunak, dan sebagai bukti pembangunan dan hasil analisa fungsionalitas perangkat lunak itu sendiri [3].



Gambar 2.2.1 Tahap proses Dokumentasi yang baik [3]

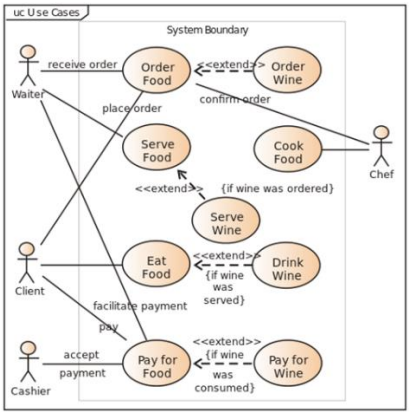
Dokumen ini berperan penting dalam rekayasa perangkat lunak karena dokumen ini menjadi salah satu faktor keberhasilan perangkat lunak yang dibangun [3] dan dokumen ini menjadi sebuah alat untuk perencanaan dan pengambilan keputusan pada pembangunan perangkat lunak [3].

2.3 Unified Modelling Language

Unified Modelling Language, atau yang biasa disingkat UML adalah notasi berbasis grafis, yang dibuat oleh Object Management Group sebagai standarisasi untuk mendeskripsikan desain berbasis perangkat lunak [8]. Notasi ini terdapat beberapa macam diagram, yang memungkinkan penjelasan dan pedeskripsian keragaman property dan aspek desain sebuah perangkat lunak [8]. Diagram yang dibuat harus dilengkapi dengan deskripsi tekstual dan lainnya untuk menghasilkan model yang lengkap. Beberapa macam diagram yang terdapat pada UML adalah *use case diagram, class diagram* dan *sequence diagram*.

2.3.1 Use Case Diagram

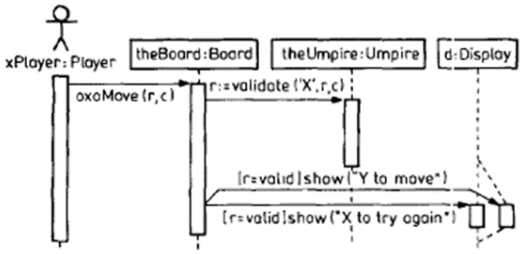
Use case diagram adalah diagram representasi sederhana interaksi pengguna dengan sebuah sistem yang menunjukan hubungan antara pengguna dan aksi yang melibatkan pengguna tersebut [8]. *Use case diagram* dapat mengidentifikasi tipe berbagai macam pengguna dari suatu sistem dan berbagai macam aksi.



Gambar 2.3.1.1 Contoh dari Use Case Diagram

2.3.2 Sequence Diagram

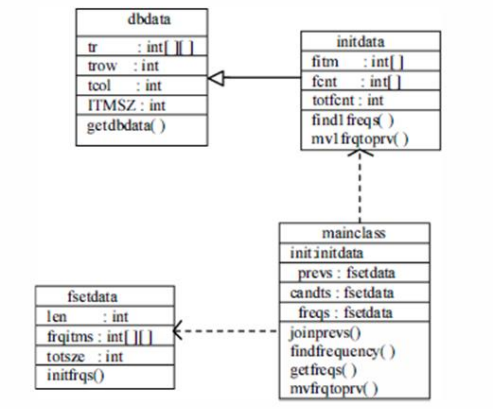
Sequence diagram membantu untuk pengertian alur eksekusi dengan urutan sekuensial dan menunjukan fitur dinamis dari sistem [9]. Objek-objek yang terlibat dan pesan-pesan yang dikirim antar objek dan urutan waktu aktivitas sistem dapat dijelaskan melalui *sequence diagram* [9].



Gambar 2.3.2.1 Contoh dari Sequence Diagram [8]

2.3.3 Class Diagram

Class diagram adalah sebuah tipe struktur diagram statis yang mendeskripsikan struktur sebuah sistem dengan cara menunjukan kelas, atribut, methods dan hubungan antar objek dari sistem tersebut [9]. *Class diagram* adalah dasar dari pemodelan berbasis objek. Diagram ini digunakan sebagai pemodelan konseptual secara umum struktur aplikasi dan sebagai penerjemahan model ke kode program [9].



Gambar 2.3.3.1 Contoh dari Class Diagram [9]

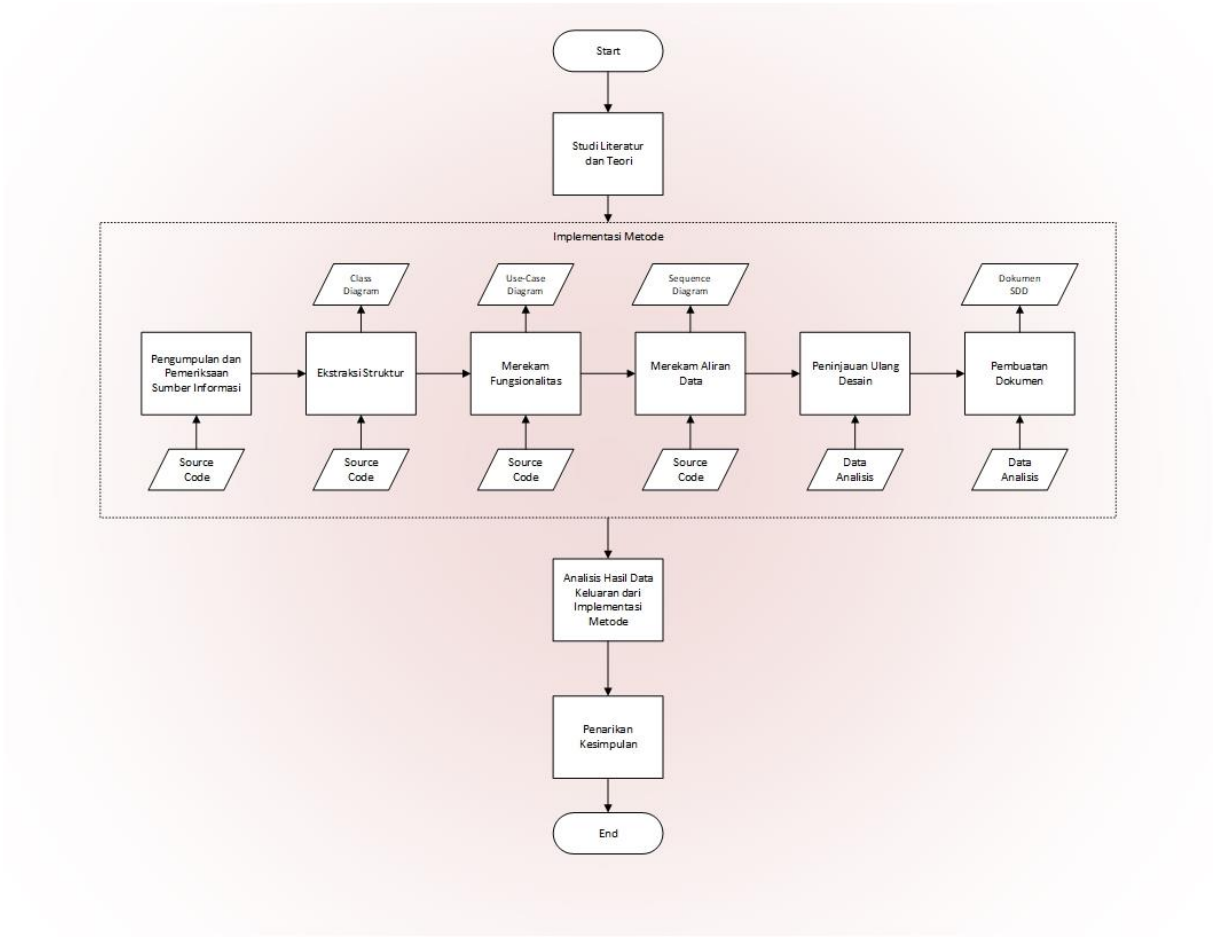
2.4 Grounded Theory

Grounded theory adalah metodologi sistematis dalam ilmu sosial yang melibatkan konstruksi teori melalui pengumpulan metodis dan analisis data [10]. Metodologi penelitian ini menggunakan penalaran induktif, berbeda dengan model hipotetis-deduktif dari metode ilmiah [10]. Sebuah studi

yang menggunakan grounded theory kemungkinan akan dimulai dengan pertanyaan, atau bahkan hanya dengan pengumpulan data kualitatif. Ketika peneliti meninjau data yang dikumpulkan, ide-ide, konsep atau elemen yang berulang menjadi jelas, dan ditandai dengan kode, yang telah diekstraksi dari data. Ketika lebih banyak data dikumpulkan, dan ditinjau kembali, kode dapat dikelompokkan ke dalam konsep, dan kemudian ke dalam kategori. Secara lebih rinci, ada 3 tahap analisis data dalam proses *grounded theory* [14]. Berikut adalah tahap-tahapnya:

1. *Open coding* : ini melibatkan pengodean baris per baris di mana konsep dan frasa kunci diidentifikasi dan disorot dan pindah ke subkategori, lalu kategori. Ini memecah data menjadi konseptual komponen dan peneliti dapat mulai berteori atau merenungkan apa yang mereka baca dan pahami membuat pengertian data. Data dari masing-masing peserta akan 'terus dibandingkan' untuk kesamaan.
2. *Axial coding* : pada tahap ini hubungan diidentifikasi antara kategori, dan koneksi diidentifikasi.
3. *Selective coding* : ini melibatkan pengidentifikasian kategori inti dan secara sistematis menghubungkannya dengan kategori lain. Hubungan harus diautentikasi dan kategori disempurnakan. Kategori kemudian diintegrasikan bersama dan GT diidentifikasi.

3. Metode Penelitian



Gambar 3.1 Flow Chart Runtunan Kegiatan Penelitian

Pada penelitian ini penulis akan membuat dokumentasi desain dari perangkat lunak yang belum memiliki dokumentasi desain apapun selama pembuatan perangkat lunak itu berlangsung sebelumnya. Ada beberapa tahap yang akan dilakukan penulis dalam penelitian ini.

3.1 Tahap Studi Teori dan Literatur

Pada tahap ini dilakukan studi teori pendukung untuk melakukan penelitian ini dan *literature review* berdasarkan *paper* yang disediakan. Teori dan *paper* yang digunakan adalah *paper* yang tersedia

di perpustakaan online yang ada di internet seperti IEEE, ScienceDirect, dan lainnya. Hal ini bertujuan untuk sebagai dasar teori yang akan digunakan selama penelitian berlangsung.

3.2 Tahap Implementasi Metode

Pada tahap ini dilakukan implementasi metode utama yaitu metode *Reverse Engineering* untuk mendapatkan desain dari perangkat lunak yang dijadikan studi kasus pada penelitian ini. Beberapa kegiatan yang dilakukan pada implementasi metode yaitu:

3.2.1 Analisis *Source Code* Perangkat Lunak

Kegiatan ini berupa pembacaan dan analisis alur dari *source code* perangkat lunak yang ada. Pembacaan *souce code* dilakukan melalui *Integrated Development Evironment* khusus untuk pembuatan aplikasi *Android* yaitu *Android Studio*. Hal ini dilakukan sebagai dasar perekaman desain dan alur data perangkat lunak yang ada.

3.2.2 Pembuatan Dokumentasi Desain

Dokumentasi desain perangkat lunak dapat dibuat setelah desain dan alur program direkam melalui tahap sebelumnya yaitu analisis *source code*. Keluaran dari proses ini adalah dokumen SDD (*Software Design Document*) sederhana yang berisi *class diagram, use-case diagram* dan *sequence diagram*. Dokumen ini berfungsi sebagai alat utama dalam modifikasi perangkat lunak yang berisi desain dan alur program.

3.2.3 Peninjauan Ulang Dokumentasi

Setelah dokumentasi dibuat, dilakukan proses peninjauan ulang dokumentasi yang berdasarkan dari *source code* perangkat lunak untuk memastikan akurasi data yang dihasilkan apakah akurat atau tidak.

3.3 Tahap Analisis Hasil Implementasi Metode dan Penarikan Kesimpulan

Pada tahap ini dilakukan analisis terhadap hasil keluaran proses metode yang dilakukan yaitu dokumentasi desain perangkat lunak. Hal ini dilakukan untuk mengetahui permasalahan yang timbul dan efektivitas metode *reverse engineering*. Hasil analisis permasalahan yang timbul didapatkan dengan melakukan analisis dengan metode *grounded theory* yang diimplementasikan kepada data *log activity* selama penelitian dan proses implementasi metode *reverse engineering* berlangsung. Serta untuk tingkat akurasi dokumentasi didapatkan dengan pembuatan ulang beberapa fitur yang ada pada diagram *use-case* dan dibandingkan dengan perangkat lunak yang ada sebelumnya. Dan pada tahap akhir adalah penarikan kesimpulan dari semua kegiatan yang berlangsung dan hasil dari analisis kegiatan dan metode selama penelitian berlangsung.

4. Evaluasi

Berdasarkan metode penelitian dan runtunan kegiatan yang telah dijelaskan dibab sebelumnya, maka diperoleh hasil dari analisis dan implementasi metode sebagai berikut:

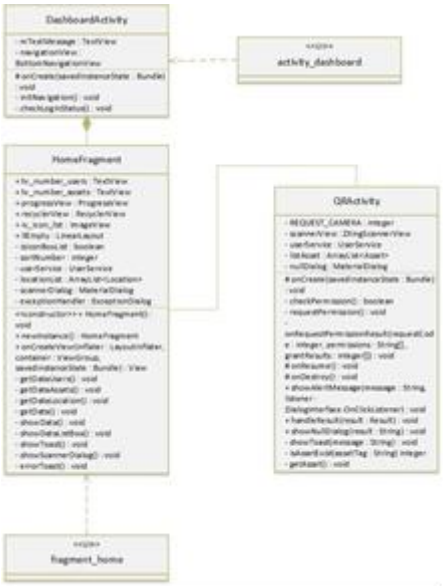
4.1 Hasil Implementasi Metode

Berdasarkan langkah-langkah yang ada di metode *Reverse Engineering*, hasil implementasi metode Reverse Engineering yang telah dilakukan pada aplikasi studi kasus yaitu berupa dokumentasi desain aplikasi yang ada. Langkah-langkah yang dilakukan pada implementasi metode adalah:

1. Pengumpulan dan Pemeriksaan Alur Informasi

   Pengumpulan informasi dilakukan melalui analisis *source code* aplikasi yang ada. Hasil dari analisis *source code* aplikasi adalah ekstraksi objek dan methods yang ada pada tiap class dari *source code* aplikasi. Hal ini dilakukan sebagai dasar data untuk pembuatan dokumen desain program. Setelah data diekstraksi langkah selanjutnya adalah pemeriksaan alur aplikasi. Hal ini dilakukan untuk mendapatkan informasi alur jalannya program baik itu alur jalannya data, aktivitas berjalannya program, dll yang juga akan dijadikan sebagai dasar pembuatan dokumen desain program.
2. Mengekstraksi Struktur

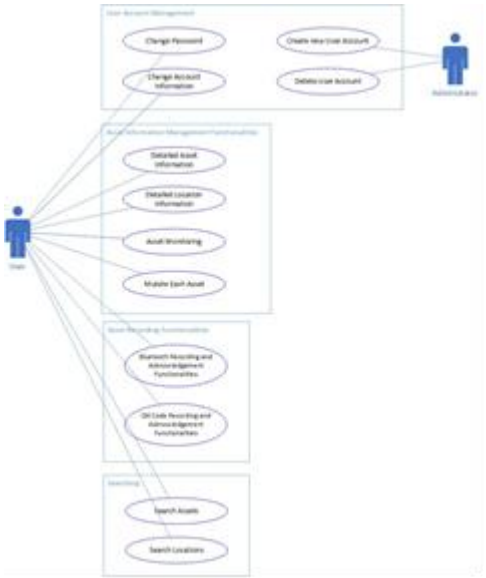
   Berdasarkan data yang didapatkan dari proses sebelumnya, didapatkan objek dan methods yang ada ditiap kelas dari *source code* aplikasi yang ada. Data objek dan *methods* menjadi dasar pembuatan *class diagram*. *Class diagram* meliputi objek (variabel) dan fungsi (*methods*) yang ada pada tiap kelas yang tersedia pada *source code* aplikasi tersebut.



Gambar 4.1.1. Contoh Salah Satu *Class Diagram* Hasil Implementasi Metode

3. Merekam Fungsionalitas

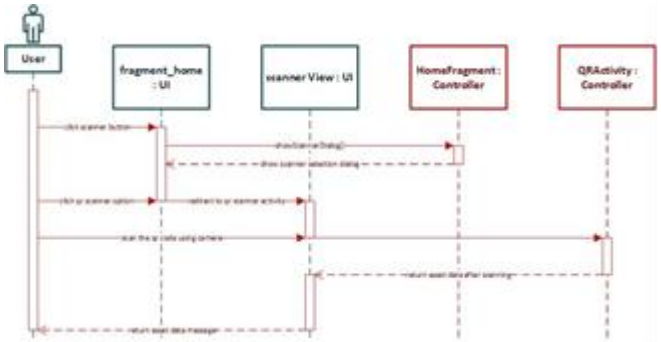
Fungsionalitas pada aplikasi adalah berupa berbagai macam aktivitas yang bisa dilakukan oleh user dan aplikasi untuk mencapai suatu tujuan tertentu. Hasil dari data fungsionalitas yang telah diektraksi menjadi dasar data pembuatan *use-case diagram. Use-case diagram* berfungsi sebagai informasi fungsi aplikasi yang dapat dimodifikasi pada versi aplikasi selanjutnya, baik itu berupa penghapusan, penambahan, perubahan, dll.



Gambar 4.1.3. *Use-Case Diagram* Hasil Implementasi Metode

4. Merekam Aliran Data

Aliran data yang dimaksud dapat berupa alur jalannya aplikasi pada tiap fungsionalitas dan alur aliran data yang diproses pada aplikasi tersebut. Hasil dari data alur aplikasi ini dijadikan dasar pembuatan *sequence diagram. Sequence diagram* berfungsi sebagai dasar data alur berjalannya tiap fungsi atau aktivitas yang ada pada aplikasi tersebut. Hal ini berupa kelas yang terlibat pada suatu fungsi, hubungan antar kelas yang terlibat pada suatu fungsi, dll.



Gambar 4.1.2. Contoh Salah Satu *Sequence Diagram* Hasil Implementasi Metode

5. Peninjauan Ulang Desain dan Pembuatan Dokumen

   Sebelum dokumen dibuat dan digunakan, peninjauan ulang data hasil analisis source code dilakukan terlebih dahulu. Hal ini dilakukan dengan cara menganalisis ulang dan pembandingan data yang dihasilkan dengan source code aplikasi untuk memastikan apakah data yang dihasilkan dari analisis sudah cukup akurat atau tidak. Setelah data yang dihasilkan sama dengan source code yang ada, pembuatan dokumen dapat dilakukan. Dokumen yang dimaksud dan dihasilkan adalah berupa prototipe dokumen SDD (*Software Design Document*) yang akan dijadikan sebagai dasar informasi modifikasi aplikasi di masa depan.

4.2 Analisis Permasalahan Yang Timbul

Selama pengerjaan pembuatan dokumentasi dari studi kasus yang ada, penulis merekam semua detil kegiatan yang berlangsung dengan cara pencatatan pada buku, mulai dari tanggal pengerjaan, hasil capaian, langkah selanjutnya, dan masalah yang dihadapi tiap kegiatan. Hal ini dilakukan sebagai pengumpulan data untuk dianalisis menggunakan metode *grounded theory* yang dijelaskan dibagian sebelumnya. Pada proses *open coding* penulis mengumpulkan beberapa data dan tema yang penulis bagi menjadi 2 tema utama yaitu *diagram making* dan *obstacles/difficulties*. Setelah itu penulis membuat tabel yang berisikan apa saja masalah yang dihadapi tiap pembuatan diagram dilakukan. Hal ini dilakukan untuk membantu mengambil kesimpulan hasil tingkat efisiensi pembuatan dokumentasi.

Gambar 4.2.1. Hasil pengelompokan pengumpulan data kegiatan

Name	Time Needed	Obstacle Found
Class Diagram	80 minutes	-
Use-Case Diagram	10 minutes	-
Sequence Diagram	160 minutes	Understanding and Theoritical issues, tricky and confusing, assistance sometimes needed

Tabel 4.2.1. Tabel detil tiap diagram yang dibuat

Berdasarkan Tabel 4.2.1 dan Gambar 4.2.1, pembuatan *class diagram* dan *use-case diagram* cenderung efisien karena tidak adanya hambatan pada proses pembuatannya. Hanya saja pada pembuatan *sequence diagram* terdapat beberapa hambatan pada proses pembuatannya yaitu kurangnya pemahaman teori yang membuat proses pembuatan diagram menjadi rumit dan membingungkan. Akibatnya, proses tersebut membutuhkan waktu yang lebih karena butuhnya pendalaman teori lebih lanjut dan bantuan dari orang yang lebih ahli. Walau secara umum proses dapat dikatakan cukup efisien, kurangnya pemahaman teori atau pengerjaan yang dilakukan oleh orang awam dapat membuat proses dokumentasi kurang bahkan tidak efisien.

Setelah dokumen dibuat, penulis melakukan beberapa percobaan untuk menganalisis tingkat efisiensi penggunaan dokumentasi desain untuk melakukan modifikasi terhadap aplikasi studi kasus. Penulis melakukan 4 percobaan yang melibatkan 4 *use-case* atau fitur yang berbeda.

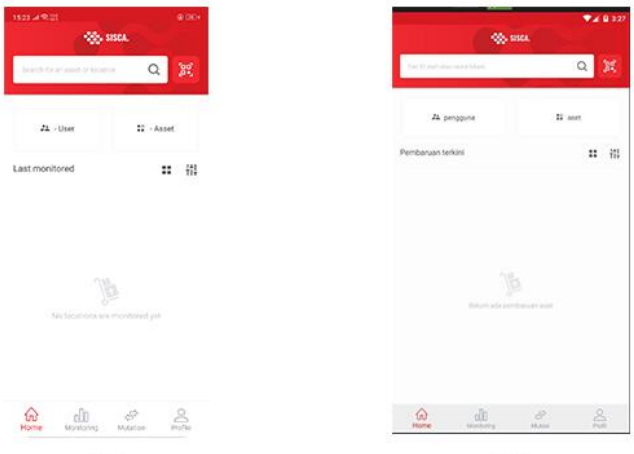
Feature Names	Modification Type	With Documentation	Without Documentation	Difference
Mutate Each Asset	Removal	25 minutes 54 seconds	38 minutes 4 seconds	31.57 %
Bluetooth Recording and Acknowledgement	Removal	9 minutes 36 seconds	21 minutes 29 seconds	57.14 %
Asset Monitoring	Removal	9 minutes 6 seconds	11 minutes 54 seconds	18.18 %
Search Asset	Updating	16 minutes 13 seconds	28 minutes 34 seconds	42.85 %
Difference Average				37.43 %

Tabel 4.2.2. Tabel perbandingan waktu yang dibutuhkan untuk menghapus fitur

Tabel 4.2.2 menunjukan hasil dari waktu yang dibutuhkan untuk menghapus dan pengupdatean beberapa fitur yang ada di aplikasi sebagai contoh modifikasi yang dilakukan terhadap aplikasi. Hasil rata-rata dari perbedaan waktu modifikasi fitur aplikasi yang dibutuhkan antara yang menggunakan dokumentasi dan yang tidak adalah 37.43%, dengan percobaan yang menggunakan dokumentasi desain membutuhkan waktu yang lebih cepat daripada yang hanya mengandalkan *source code* saja. Hal ini dikarenakan dokumentasi desain membantu penulis program untuk memahami alur desain aplikasi yang dimodifikasi dengan lebih mudah. Sehingga perubahan yang dilakukan dapat lebih maksimal dan efisien. Hanya saja percobaan yang dilakukan pada penelitian ini dilakukan oleh penulis juga sehingga dapat menimbulkan bias pada data hasil percobaan.

4.3 Analisis Akurasi Data

Setelah dokumen yang dihasilkan setelah melakukan proses *reverse engineering* selesai, penulis memvalidasi tingkat keakuratan data yang dihasilkan oleh metode tersebut. Pada tahap ini, penulis melakukan penulisan program ulang yang bertuju pada dokumen yang dihasilkan. Sebagai pengujian lebih lanjut, penulis melakukan perubahan fitur pada salah satu fungsionalitas pada aplikasi studi kasus. Hasil dari perubahan ini tidak menunjukan perubahan fungsi dari aplikasi yang signifikan, karena demikian proses fungsi dari aplikasi masih relatif sama dengan fungsi utama. Berbeda dengan contoh penghapusan fitur, namun hasil yang ditampilkan oleh aplikasi tetap dapat berjalan dengan baik karena tidak ada hubungan antar kode dan kelas yang rusak walau ada fitur yang dihapus.

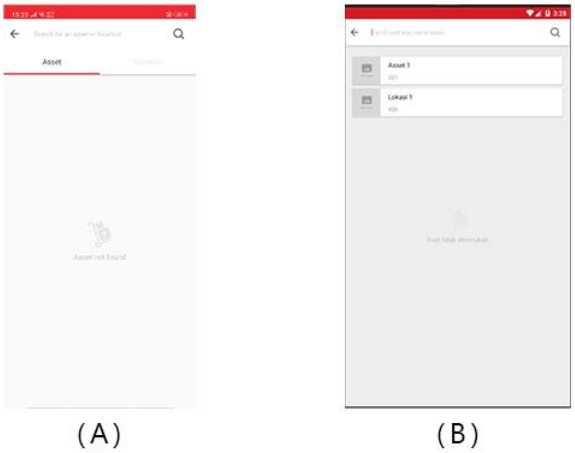


(A) (B)

Gambar 4.3.1. Contoh Hasil Penulisan Ulang Fitur

Pada gambar 4.3.1, penulis melakukan penulisan ulang salah satu fitur utama yang ada di aplikasi studi kasus. Sebagai bahan referensi, penulis bertuju pada dokumen yang ada, berfokus pada *class diagram* yang merepresentasikan hubungan antar kelas yang berkaitan, dan *sequence diagram* yang merepresentasikan alur data antar fitur. Hasil yang didapat adalah relatif sama dengan fitur dasar dari aplikasi utama.

Gambar 4.3.2. Contoh Kode Yang Dilakukan Perubahan



Gambar 4.3.3. Contoh Hasil Penulisan Ulang dan Perubahan Fitur

Pada gambar 4.3.3, penulis melakukan penulisan salah satu fitur lainnya yang ada di aplikasi dengan sedikit perubahan pada hasil akhirnya. Walau secara fitur dan tampilan menjadi sedikit berbeda pada hasil akhirnya, tetapi secara fungsi masih relatif sama dan melakukan proses yang sama. Sehingga perubahan fitur yang dihasilkan tidak begitu signifikan dan masih mengeluarkan *output* yang sama.

5. Kesimpulan

Pada penelitian ini dilakukan implementasi metode reverse engineering pada sebuah aplikasi yang belum memiliki dokumentasi apapun baik desain maupun kebutuhan. Pada penelitian ini penulis berfokus pada proses pembuatan dokumentasi desain aplikasi yang meliputi dari 3 diagram yaitu class diagram, sequence diagram dan use-case diagram. Penelitian ini menemukan bahwa metode *reverse engineering* termasuk metode yang baik untuk digunakan untuk mendapatkan data desain aplikasi yang belum memiliki dokumentasi sama sekali. Berdasarkan hasil implementasi metode ada 2 poin yang dianalisis yaitu tingkat akurasi data yang dihasilkan dan tingkat efisiensi dari metode yang digunakan. hanya saja keahlian dari penganalisa (yang berupa pemahaman secara teori dan pengalaman) sangat mempengaruhi tingkat efisiensi dari implementasi dari metode ini. Selanjutnya hasil tingkat efisiensi dari modifikasi aplikasi menggunakan dokumentasi termasuk lebih efisien dalam hal waktu dan usaha yang dilakukan, baik itu perubahan fitur dan penghapusan fitur yang ada diaplikasi tersebut. Hal ini dibuktikan dengan adanya perbedaan presentase waktu yang menunjukan modifikasi menggunakan dokumentasi membutuhkan waktu sekitar 37% lebih cepat dari modifikasi yang tidak

menggunakan dokumentasi sama sekali. Serta tingkat akurasi data yang dihasilkan dari proses implementasi metode *reverse engineering* termasuk dalam kategori cukup akurat. Hal ini dikarenakan data dari dokumen yang dihasilkan dapat memberikan informasi yang cukup dan jelas untuk penulis aplikasi pada saat implementasi dokumen desain menjadi aplikasi utuh. Serta informasi yang diberikan dapat menjadi acuan jika adanya rencana perubahan fitur yang akan ditambah pada versi aplikasi selanjutnya.

Daftar Pustaka

[1] H. Muccini, A. D. Francesco and P. Esposito, "Software Testing of Mobile Applications: Challenges and Future Research Directions," p. 7, 2012.
[2] P. Laplante, What Every Engineer Should Know about Software Engineering, Boca Raton: CRC, 2007.
[3] N. J. Kipyegen and W. P. K. Korir, "Importance of Software Documentation," vol. 10, no. 5, p. 6, 2013.
[4] D. L. Parnas, in The Future of Software Engineering, Heidelberg, Springer, 2011, pp. 125-148.
[5] E. J. Byne, "Software Reverse Engineering: A Case Study," p. 16, 1991.
[6] J. Thankappan and V. Patil, "Detection of Web Design Patterns Using Reverse Engineering," p. 5, 2015.
[7] A. Jain, S. Soner and A. Gadwal, "Reverse Engineering: Journey from Code to Design," p. 5, 2011.
[8] P. Pooley and P. King, "The Unified Modelling Language and Performance Engineering," vol. 146, no. 1, p. 9, 1999.
[9] R. Rathinasabapathy, "Object oriented software design for association rule mining algorithms using sequence diagram," p. 4, 2015.
[10] Y. C. Tie, M. Birks and K. Francis, "Grounded theory research: A design framework for novice researchers," vol. 7, p. 8, 2019.
[11] P. Tripathy and K. Naik, Software Evolution and Maintenance, New Jersey: Wiley, 2015.
[12] E. J. Chikofsky and J. H. Cross II, "Reverse Engineering and Design Recovery: A Taxonomy," p. 5, 1990.
[13] N. Shi and R. A. Olsson, "Reverse Engineering of Design Patterns from Java Source Code," p. 10, 2006.
[14] H. Noble and G. Mitchell, "What Is Grounded Theory?," p. 3, 2016.