REPOSITOR, Vol. 6, No. 1, Februari 2024, Pp. 1-10
ISSN : 2714-7975
E-ISSN : 2716-1382

Penerapan Desain Pattern Observer Pada Pengembangan Aplikasi Android (Studi Kasus: Aplikasi KataFilm)

Ilham Hadisyah Ramadhani*[1], Wildan Suharso[2], Didih Rizki[3]
[1,2,3]Universitas Muhammadiyah Malang
ilhamhadisyahsmd@gmail.com*

Abstrak

Aplikasi seluler sangat mempengaruhi hampir di semua bidang terpenting di kehidupan seperti perbankan, kesehatan, militer dan masih banyak lagi. Dengan hadirnya aplikasi-aplikasi tersebut membawa pengaruh besar di era digital ini terutama perubahan pada masyarakat, baik budaya, etika dan norma yang ada. Dengan keadaan tersebut, banyak perusahaan IT yang dituntut untuk mengembangkan aplikasi seluler dengan cepat, namun mengesampingkan kualitas kode yang menyebabkan meningkatnya kompleksitas kode dan menurunkan nilai metrix maintainability-nya. Salah satu cara untuk menangani permasalahan yang diuraikan diatas adalah penggunaan pola desain yang tepat, salah satunya adalah pola desain observer. Tujuan penelitian ini adalah mengimplementasikan pola desain observer pada aplikasi KataFilm yang sebelumnya memiliki desain pattern Chain of Responsibility yang dianggap memiliki kekurangan khususnya pada sisi maintainabilitynya.

Kata Kunci: *Pola Desain, Observer, Chain of Responsibility*

Abstract

Mobile applications greatly influence almost all the most important areas of life such as banking, health, payments and many more. The presence of these applications has had a big influence in this digital era, especially changes in society, both culture, ethics and existing norms. With this situation, many IT companies are required to develop mobile applications quickly, but neglect the quality of the code that causes the complexity of the code to increase and reduces the maintainability metric value. One way to handle the problems described above is to use appropriate design patterns, one of which is the observer design pattern. The aim of this research is to implement a pattern observer design in the KataFilm application which previously had a Chain of Responsibility pattern design which was considered to have shortcomings, especially in terms of maintainability.

Keywords: *Design Pattern, Observer, Chain of Responsibility*

1. Pendahuluan

Aplikasi seluler sangat mempengaruhi hampir di semua bidang terpenting di kehidupan seperti perbankan, kesehatan, pembayaran, militer dan masih banyak lagi. Dengan hadirnya aplikasi-aplikasi tersebut membawa pengaruh besar di era digital ini terutama perubahan pada masyarakat, baik budaya, etika dan norma yang ada. Terlebih lagi, banyak perusahan IT yang berlomba-lomba untuk menciptakan banyak aplikasi yang diharapkan mampu memenuhi kebutuhan masyarakat. Sehingga, banyak perusahaan yang dituntut untuk segera menyelesaikan aplikasinya dengan cepat sehingga dapat mengurangi biaya perancangan atau pengembangan aplikasi. Dalam implementasinya, beberapa aplikasi bahkan dari perusahaan yang besar semacam Google pernah menghapus beberapa aplikasinya yang tidak di maintain dalam kurun waktu tertentu [1][2]. Hal tersebut disebabkan oleh buruknya desain aplikasi yang dimilikinya sehingga, pada saat tahapan maintenance atau pengembangan, pengembang kesulitan untuk memeliharanya.

Dalam mengembangkan dan memelihara aplikasi seluler, peran *Design Pattern* (Pola Desain) dianggap membantu bagi pengembang aplikasi. Pentingnya penerapan pola desain terletak pada kemampuannya untuk membantu pengembang agar mencapai kode yang lebih bersih, lebih terstruktur, lebih mudah dikelola, serta menghemat waktu dan sumber daya dalam siklus pengembangan perangkat lunak. *Design pattern* merupakan pedoman atau *template* yang memberikan struktur dan kerangka kerja yang teruji dan terbukti efektif dalam menghadapi jenis

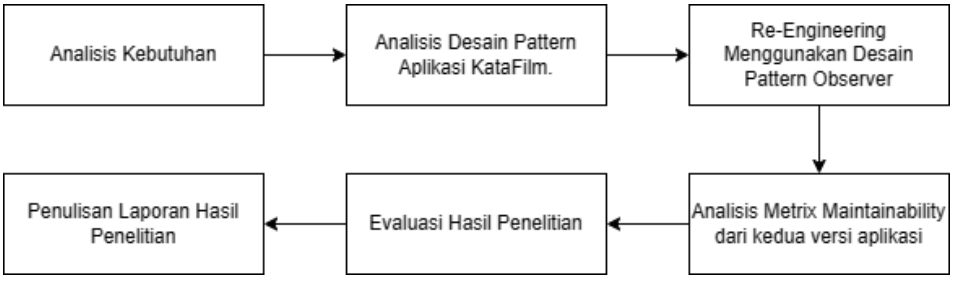
masalah tertentu. Dalam konteks pengembangan perangkat lunak, *design pattern* adalah solusi umum untuk mempercepat proses pengembangan serta menanggapi masalah yang sering muncul dalam implementasi aplikasi salah satunya kualitas kode yang dianggap kurang baik. Kualitas kode yang baik menjadi elemen penting dalam pengembangan perangkat lunak yang berkelanjutan untuk memastikan bahwa aplikasi dapat dikelola dengan baik, minim kesalahan, dan mampu berkembang seiring waktu. Oleh karena itu, penting untuk menjaga kualitas kode aplikasi melalui penggunaan Design Pattern yang lebih baik[3].

Pada penelitian ini, mengambil studi kasus aplikasi KataFilm yaitu aplikasi portofolio yang pernah peneliti kembangkan, merupakan sebuah aplikasi katalog film yang memiliki fitur rating dan share. Dalam pengembangan aplikasi KataFilm, sebelumnya menggunakan pola desain *Chain of Responsibility* (CoR) yang di klasifikasikan berdasarkan pada bagaimana cara data yang telah di dapat dari API *Service* di teruskan ke layer UI. Beberapa tahun setelah aplikasi tersebut selesai di kembangkan, peneliti menemukan sebuah kesulitan pada saat pengembangan yang dilakukan. Menurut Manh Phan, penerapan desain pattern CoR menimbulkan kelemahan di sisi maintainability-nya dimana timbulnya banyak *boilerplate code* (kode yang ditulis berulang) yang dihasilkan jika aplikasi yang di implementasi memiliki skala yang besar [4]. *Boilerplate code* yang berlebihan dapat membuat aplikasi menjadi sulit untuk dikembangkan dan memakan waktu. Ini memperjelas pentingnya penggunaan design pattern yang tepat untuk menghindari atau mengurangi *boilerplate code*, sehingga pengembangan aplikasi dapat menjadi lebih efisien dan mudah dikelola. Dalam rangka meningkatkan kualitas kode dan mengatasi masalah yang telah diidentifikasi, penelitian ini akan fokus pada penerapan *design pattern* lain yang dianggap lebih sesuai untuk meningkatkan maintainability dan efisiensi dalam pengembangan aplikasi KataFilm.

Design pattern yang akan diadopsi dalam penelitian ini adalah *Design Pattern Observer*. *Observer* memungkinkan pemisahan antara objek yang menghasilkan perubahan (*subject*) dengan objek-objek yang harus merespons perubahan tersebut (*observers*), sehingga mengurangi ketergantungan antar komponen dalam aplikasi [10]. Dengan menggunakan *Observer*, diharapkan aplikasi KataFilm akan menjadi lebih fleksibel dan mudah diubah serta mengurangi *boilerplate code*, yang pada gilirannya akan meningkatkan kualitas kode dan *maintainability*-nya. *Design pattern observer* memiliki kelebihan dalam sisi kualitas kode dan meminimalisir munculnya *bug* (kegagalan sistem) [5]. Pada kasus ini, penerapan dilakukan dengan cara melakukan *refactor* pada kode yang sudah ada pada aplikasi KataFilm. Setelah melalui proses *refactoring*, kedua versi aplikasi tersebut akan dihitung nilai metrix *maintainability*-nya dan membandingkan hasil dari metrix yang diperoleh setelah melalui proses refactor. Harapan pada penelitan ini adalah membuktikan apakah *design pattern observer* memiliki nilai *maintainability* yang lebih baik dan dapat meningkatkan metrik *maintainability* pada sebuah aplikasi daripada *design pattern Chain of Responsibility*.

2. Metode Penelitian

Beberapa tahapan penelitian yang dilakukan untuk mendapatkan nilai metrik dari aplikasi KataFilm di kedua versi *design pattern* nya dapat dilihat pada Gambar 1



Gambar 1 Tahapan Penelitian

2.1 Analisis Kebutuhan.

Pada tahap ini, peneliti akan melakukan identifikasi masalah yang terjadi pada desain pattern yang telah terimplementasi pada aplikasi KataFilm yang telah peneliti kembangkan. Dalam implementasinya, aplikasi KataFilm menggunakan desain pattern *Chain of Responsibility* (CoR)*,* yang dimana pada desain pattern tersebut memiliki kelemahan yakni salah satunya adalah menimbulkan banyaknya boilerplate [4]. Sehingga dengan kelemahan desain pattern CoR,

peneliti akan melakukan refaktoring pada desain pattern *Chain of Responsibility* pada aplikasi tersebut menjadi desain pattern Observer.

2.2 Analisis Desain Pattern Aplikasi KataFilm.

Analisa design pattern yang terdapat pada aplikasi KataFilm dilakukan dengan cara menghitung nilai maintainability-nya. Pada dasarnya, *maintainability* memfokuskan pada kemampuan sistem perangkat lunak pada fase maintenance atau perbaikan dalam menerima suatu perubahan [6]. Dalam bukunya yang berjudul "*Practical Reability Engineering*", Patrick menjelaskan bahwa *maintainability* mempengaruhi tingkat availability secara langsung. Dimana waktu pada *maintainability* tersebut diambil untuk fokus memperbaiki kerusakan secara rutin untuk mengambil nilai state yang tersedia (*available state*). Sehingga dapat dikatakan bahwa *maintainability* dan *reability* memiliki hubungan yang erat, karena yang satu mempengaruhi yang lain dan kedua-duanya mempengaruhi *availability* (ketersediaan) dan *cost* (biaya) yang ada.

Sebelum menghitung nilai *maintainability* pada aplikasi KataFilm, terlebih dahulu dilakukan pengumpulan data berupa:

1. Kode sumber (*source code*) dari aplikasi KataFilm.
2. Informasi tentang bahasa pemrograman, platform, dan teknologi yang digunakan dalam pengembangan aplikasi KataFilm.
3. Evaluasi penggunaan *Design Pattern Observer* pada aplikasi KataFilm.
4. Dokumentasi dari alat analisis kode (*code analysis tool*) yang digunakan untuk melakukan analisis pada aplikasi KataFilm.
5. Data kinerja (*performance*) dan keandalan (*reliability*) dari aplikasi KataFilm, seperti laporan waktu respon, laporan error, dan laporan performa aplikasi.

Setelah melakukan pegumpulan data, data tersebut akan disimpan dan dijadikan acuan untuk membandingkan apakah *Design Pattern Observer* memecahkan masalah maintainability pada sebuah projek aplikasi android.

2.3 Re-Engineering Menggunakan Desain Pattern Observer

Setelah melakukan analisis terhadap masalah pada desain pattern yang digunakan pada aplikasi KataFilm, peneliti dapat mengusulkan beberapa perbaikan untuk *re-engineering* desain pattern tersebut. Berikut adalah beberapa perbaikan yang dapat dilakukan:

1. Menggunakan MVP (*Model-View-Presenter*) pattern

   MVP pattern merupakan salah satu desain pattern yang dapat digunakan untuk memisahkan antara logika bisnis (*presenter*) dan tampilan (*view*) pada sebuah aplikasi. Dengan menggunakan MVP pattern, maka perubahan yang terjadi pada tampilan tidak akan berdampak pada logika bisnis [7]. Selain itu, MVP pattern juga dapat mempermudah pengujian dan memungkinkan untuk membuat beberapa *view* yang berbeda tanpa harus mengubah logika bisnis yang ada.
2. Menggunakan *Dependency Injection*

   Dependency Injection adalah sebuah teknik dalam pemrograman yang digunakan untuk memudahkan pengelolaan dependensi antara kelas-kelas yang ada dalam sebuah aplikasi. Dengan menggunakan *Dependency Injection*, maka kita dapat memisahkan antara pembuatan objek dengan penggunaan objek tersebut sehingga mempermudah dalam pengujian dan memungkinkan untuk melakukan substitusi objek dengan objek yang lain [8].
3. Menggunakan *Repository Pattern*

   Repository Pattern digunakan untuk memisahkan antara logika akses data dengan logika bisnis pada sebuah aplikasi. Dengan menggunakan *Repository Pattern*, maka kita dapat memisahkan antara akses data dengan logika bisnis sehingga memudahkan dalam pengujian dan memungkinkan untuk melakukan substitusi repositori dengan repositori yang lain.
4. Menggunakan Observer Pattern Untuk Mengakses Data Film

   Aplikasi KataFilm menggunakan API dari *The Movie Database* (TMDb) sebagai sumber data film yang disediakan. Untuk mengakses data film dari REST API TMDb, digunakan *Observer Pattern* dalam implementasinya. Terdapat dua *Observer* yang mengakses data film yaitu 'FilmListPresenter' dan 'FilmDetailPresenter'. 'FilmListPresenter' digunakan untuk mengakses daftar film yang ditampilkan pada halaman utama aplikasi.

Sedangkan ‘FilmDetailPresenter’ digunakan untuk mengakses detail dari film yang dipilih oleh pengguna.

2.4 Analisis Metrix Maintainability Dari kedua Versi Aplikasi

Untuk melakukan analisa maintainability pada aplikasi KataFilm, terdapat beberapa faktor yang dapat dipertimbangkan, seperti kualitas kode, kemudahan dalam melakukan perubahan, serta dokumentasi dan struktur aplikasi. Kualitas kode merupakan aspek penting yang melibatkan readability, maintainability, dan kompleksitas kode [9]. Kode yang baik harus mudah dibaca, dimengerti, dan mudah dipelihara oleh programmer lain. Pada aplikasi KataFilm, ditemukan beberapa ketidak-konsistenan, seperti penggunaan nama variabel yang berbeda pada kelas yang sama dan penggunaan tipe data yang tidak tepat pada beberapa function.

Selain itu, beberapa metode juga memiliki kompleksitas yang tinggi, seperti pada fungsi ‘searchMovie’ pada kelas ‘SearchViewModel’ yang melakukan beberapa operasi pencarian pada list. Hal ini dapat mempersulit pemeliharaan kode dan perubahan di masa depan. Faktor kedua yang dapat dipertimbangkan adalah kemudahan dalam melakukan perubahan pada aplikasi. Meskipun telah diterapkan *Design Pattern Observer* untuk memudahkan perubahan, terdapat beberapa kelas yang memiliki tanggung jawab yang terlalu banyak, seperti pada class ‘MovieViewModel’ yang bertanggung jawab pada logika bisnis serta komunikasi antar-class.

Hal ini dapat menyulitkan dalam melakukan perubahan pada salah satu aspek tanpa mempengaruhi aspek lain. Selain itu, faktor ketiga yang dapat dipertimbangkan adalah dokumentasi dan struktur aplikasi. Pada aplikasi KataFilm, dokumentasi yang disediakan masih terbatas pada beberapa komentar kode dan tidak tersedia dokumentasi teknis secara terstruktur. Hal ini dapat menyulitkan programmer lain dalam memahami kode dan melakukan perubahan pada aplikasi. Struktur aplikasi juga perlu diperhatikan untuk memudahkan programmer dalam menemukan dan memodifikasi kode yang diperlukan.

2.5 Evaluasi Hasil Penelitian

Evaluasi hasil penelitian dilakukan menggunakan metode kuantitatif, yakni melakukan evaluasi dari hasil *re-engineering* desain pattern yang sudah diimplementasikan pada aplikasi KataFIlm sehingga dapat mengetahui perbandingan dari kedua desain pattern tersebut. Selanjutnya, evaluasi hasil analisis maintainability pada aplikasi KataFilm setelah menggunakan desain pattern *Observer* dengan melihat hasil nilai *debt* dan *debt ratio* yang ditunjukkan pada SonarQube. Dari hasil evaluasi tersebut sehingga dapat disajikan rekomendasi untuk penelitian selanjutnya terhadap desain pattern *Observer*.

3. Hasil Penelitian dan Pembahasan

Dari tahapan penelitian yang sudah dibahas sebelumnya, maka didapat hasil penelitian dari pembahasan yang akan dijabarkan pada sub-bab berikut:

3.1 Hasil Analisis kebutuhan

Setelah dilakukan identifiaksi masalah, desain pattern CoR yang ter-implementasi pada aplikasi KataFim ternyata memiliki beberapa masalah dalam pengimplementasian-nya. Salah satu masalahnya adalah hierarki yang terlalu kaku, di mana setiap *handler* (penangan) dalam rantai harus menangani permintaan atau meneruskannya, tanpa memberikan fleksibilitas yang cukup. Selain itu, tanggung jawab setiap *handler* terkadang tidak jelas, menyebabkan kebingungan dalam menentukan *handler* yang seharusnya menangani jenis permintaan tertentu. Kinerja juga menjadi perhatian, terutama dalam rantai panjang yang dapat mengakibatkan pemborosan kode yang signifikan.

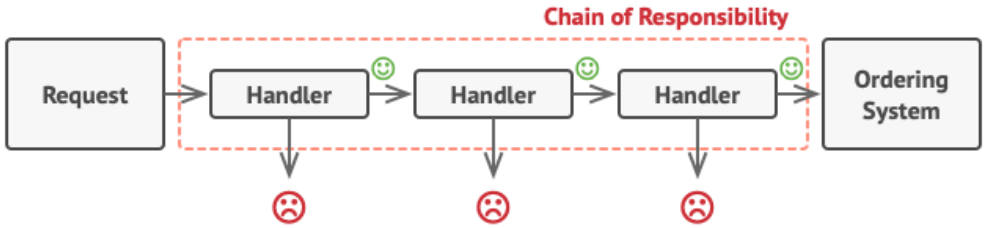
Dari permasalahan tersebut, refaktor menjadi kebutuhan mendasar untuk meningkatkan kualitas dan fleksibilitas sistem pada aplikasi KataFilm. Hierarki yang terlalu kaku memicu upaya refaktor untuk memberikan fleksibilitas yang lebih besar dalam menangani permintaan, memperjelas tanggung jawab *handler*, memudahkan perubahan dinamis pada rantai *handler*, meningkatkan kinerja sistem, dan memperbaiki kesulitan dalam pemeliharaan kode.

Dalam konteks refaktor ke desain pattern *Observer*, model arsitektur baru diharapkan dapat menciptakan sistem yang lebih adaptif dan responsif. Dengan memisahkan tanggung jawab antara subjek dan observer, desain pattern *Observer* memberikan fleksibilitas dan modularitas yang dibutuhkan untuk menangani perubahan dinamis dalam sistem. Keuntungan melibatkan kemampuan menangani perubahan dengan lebih dinamis, meningkatkan

keterbacaan dan pemahaman kode, serta memfasilitasi pengembangan dan pemeliharaan sistem secara lebih efektif. Sehingga, refaktor ke desain pattern *Observer* dianggap sebagai langkah yang tepat untuk meningkatkan fleksibilitas dan adaptabilitas aplikasi KataFilm.

3.2 Hasil Analisis Desain Pattern Aplikasi *KataFilm*

Pada aplikasi KataFilm, desain pattern yang ter-implementasi adalah desain pattern *Chain of Responsibility* (CoR). Desain pattern CoR adalah salah satu dari 23 desain pattern yang dikenal sebagai *behavioral design pattern*. Desain pattern ini memungkinkan kita untuk melewatkan permintaan (*request*) melalui serangkaian *handler*. Setiap *handler* akan memutuskan apakah akan memproses permintaan tersebut atau meneruskannya ke *handler* berikutnya dalam chain. Pada Gambar 2, dalam desain pattern ini, setiap handler harus memiliki satu metode yang melakukan pengecekan. Permintaan beserta datanya akan diteruskan ke metode ini sebagai argumen [7].



Gambar 2 Diagram Alur Desain Pattern Chain of Responsibility (CoR)

Dalam hasil analisa desain pattern pada aplikasi KataFilm, telah diidentifikasi potensi perubahan yang dapat meningkatkan nilai *maintainability* (kemudahan perawatan) aplikasi. Perhatian khusus diberikan pada UI layer, di mana tindakan atau aksi yang diambil oleh aplikasi bergantung pada kondisi-kondisi tertentu. Semakin banyak kondisi yang ada, semakin banyak kode yang harus ditulis untuk mengelola berbagai kemungkinan skenario. Salah satu potensi untuk mempernbaiki kelemahan tersebut adalah menggunakan desain pattern *Observer*.

Observer adalah salah satu pola desain yang sangat relevan dalam konteks aplikasi KataFilm. Dalam desain pattern ini, terdapat dua komponen utama, yaitu *Publisher* (penyedia data) dan *Subscriber* (pengamat perubahan data). Cara kerja dari Observer pada aplikasi KataFilm adalah ketika seorang pengguna menyimpan atau menghapus film dari daftar favorit mereka, *Publisher* akan menerima perubahan data ini, baik melalui interaksi pengguna atau melalui perubahan data dari jaringan atau basis data. Setelah menerima perubahan data, *Publisher* akan memberikan notifikasi kepada semua *Subscriber* yang telah mendaftar untuk mendapatkan pembaruan terkait data ini.

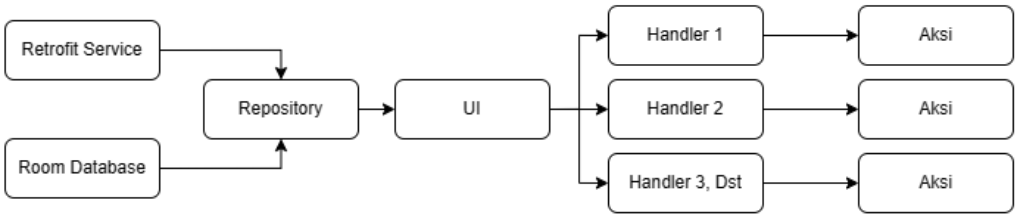
Dengan menerapkan pola desain *Observer*, aplikasi KataFilm dapat memberikan pengalaman yang lebih dinamis dan responsif kepada penggunanya. Ini memungkinkan pengguna untuk tetap terinformasi tentang perubahan data yang terkait dengan film dan acara TV yang mereka minati, sehingga meningkatkan kenyamanan dan kualitas penggunaan aplikasi.

3.3 Hasil Re-Engineering Menggunakan Desain Pattern Observer

Dalam analisa aplikasi KataFilm sebelumnya, telah diidentifikasi adanya potensi perbaikan untuk mengurangi kemungkinan penulisan kode yang berulang atau *boilerplate code*. Untuk mencapai tujuan ini, perubahan desain pattern dari CoR ke *Observer* diusulkan sebagai solusi yang lebih efisien. Dengan CoR sebelumnya, setiap kondisi memerlukan implementasi yang berbeda, yang bisa memunculkan banyak kode yang hampir serupa. Dengan mengadopsi *Observer*, kita dapat mencapai abstraksi yang lebih tinggi dan efisien, dimana *Publisher* bertanggung jawab untuk mengelola data dan memberikan notifikasi kepada *Subscriber* ketika data berubah. Ini akan mengurangi duplikasi kode dan membuat pengembangan lebih efisien.

Pada aplikasi KataFilm, desain pattern yang diimplementasikan adalah CoR. Desain ini mengadopsi pola instruktif berantai, di mana serangkaian objek penanganan (*handler*) saling terhubung. Setiap *handler* memiliki tanggung jawab khusus dalam memproses permintaan, dan jika handler tertentu tidak dapat menangani permintaan tersebut, ia akan meneruskannya ke handler berikutnya dalam rantai.

Pada implementasinya, desain pattern CoR dalam aplikasi KataFilm difokuskan pada cara efektif meneruskan data dari Retrofit dan Room Database ke antarmuka pengguna (UI). Dengan memanfaatkan rantai *handler*, setiap tahap dalam proses, mulai dari pengambilan data melalui Retrofit hingga penyimpanan dan pengambilan data dari *Room Database*, dapat diatur secara terstruktur. Setiap handler bertanggung jawab untuk memproses dan meneruskan data dengan cara yang sesuai.

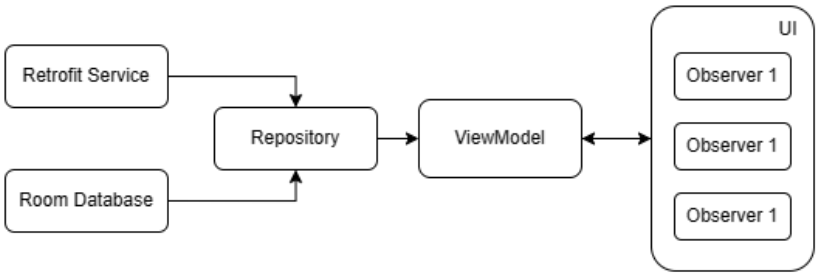


Gambar 3 Pola Desain Chain of Responsibility (CoR) pada aplikasi KataFilm

Jika kita mengamati Gambar 3, desain pattern CoR terlihat memiliki rantai yang tidak terputus, membentang mulai dari tahap awal perolehan data hingga saat data berhasil ditampilkan di antarmuka pengguna (UI). Setiap elemen dalam rantai, mulai dari pengambilan data melalui Retrofit hingga manipulasi dan penyimpanan data di *Room Database*, saling terkait dan bekerja secara terkoordinasi. Kesinambungan ini memastikan bahwa setiap tahap proses dijalankan dengan urutan yang benar, membentuk aliran data yang lancar dan terpadu hingga akhirnya mencapai tampilan yang sukses di UI.

Pada implementasi desain pattern CoR, dapat terlihat bahwa rantai yang tidak terputus akan menimbulkan *boilerplate code* karena setiap handler dalam rantai perlu mengimplementasikan metode penanganan permintaan. Hal ini dapat menyebabkan duplikasi kode yang signifikan, terutama jika tanggung jawab dasar dari setiap *handler* serupa. Dengan alasan tersebut, desain pattern *Observer* diusulkan sebagai solusi untuk mengatasi masalah duplikasi atau yang biasa disebut boilerplate dalam implementasi desain pattern Chain of Responsibility. Dalam desain observer, objek yang mengamati (*observer*) dapat merespons perubahan status tanpa memerlukan hierarki yang kompleks. Hal ini meminimalkan boilerplate code karena handler hanya perlu mendaftar atau melepaskan diri sebagai observer, mengurangi kebutuhan untuk implementasi ulang yang berlebihan dan mempermudah pemeliharaan kode.

Bagian fungsional yang akan diubah adalah bagaimana cara menyampaikan data dari repository ke UI, yaitu dengan mengganti pendekatan rantai data menjadi penggunaan publisher dan observer. Dalam konteks ini, data dari repository diumumkan sebagai sumber (*publisher*), dan komponen UI yang membutuhkan data tersebut berperan sebagai observer yang dapat merespons perubahan. Dengan mengadopsi pola desain *publisher* dan *observer*, kita dapat menciptakan hubungan yang lebih fleksibel dan longgar antara komponen-komponen, mengurangi ketergantungan langsung, serta meminimalkan *boilerplate code* yang mungkin muncul dalam desain sebelumnya.



Gambar 4 Diagram Area Refaktor Pada Aplikasi Katafilm

Jika melihat dari Gambar 4, area perubahan akan terjadi pada repositori, yang selanjutnya akan diteruskan ke UI. Dalam konteks ini, perubahan tersebut memerlukan penambahan kelas ‘ViewModel’ yang bertindak sebagai perantara. ‘ViewModel’ ini akan mengekstraksi data dari

repository dan memasangnya pada publisher, yang selanjutnya akan diobservasi oleh observer di komponen UI. Dengan adanya 'ViewModel', proses pengelolaan data menjadi lebih terstruktur dan memungkinkan pemisahan tanggung jawab antara lapisan data dan antarmuka pengguna, memberikan fleksibilitas yang lebih besar dalam pengembangan dan pemeliharaan aplikasi.

3.4 Hasil Analisis Metrix Maintainability Dari Kedua Versi Aplikasi

Analisis metrix *maintainability* dilakukan dengan menganalisis kode pada aplikasi KataFilm. Adapun *software* yang dibutuhkan yaitu JDK 11 (OpenJDK), SonarQube Versi 9.9 LTS, Windows 11 2H22, Command Line, Android Studio, serta Web Browser. Langkah yang dilakukan adalah membuat proyek di aplikasi SonarQube sesuai dengan proyek yang akan dianalisis untuk menampilkan hasil analisis kode-nya.

Selanjutnya, dilakukan integrasi proyek Android Studio dengan SonarQube. Buka file "build.gradle" dalam modul proyek Android Studio. Tambahkan plugin dengan ID "org.sonarqube" versi "3.5.0.2730" pada berkas tersebut. Kemudian, tambahkan konfigurasi SonarQube pada blok "sonarqube" dengan mengisi properti seperti "sonar.projectKey", "sonar.host.url", "sonar.login", dan "sonar.password" sesuai dengan informasi yang diperlukan. Setelah melakukan konfigurasi, buka terminal pada proyek Android Studio dan jalankan perintah "./gradlew sonar" untuk memulai proses analisis kode. Tunggu hingga proses selesai, dan hasil analisis kode akan tersedia pada proyek SonarQube yang telah dibuat sebelumnya.

Tabel 1 berikut merupakan hasil dari pengukuran maintainability aplikasi KataFilm tanpa menggunakan desain pattern observer serta menggunakan desain pattern *Observer*:

Tabel 1 Hasil Pengukuran Dengan Desain Pattern Chain of Responsibility (CoR)

Code Smells	Debt (Hour, minutes)	Debt Ratio	Rating	Effort to Reach A
15	5h 30 min	0,30%	A	0

Tabel 2 Hasil Pengukuran Dengan Desain Pattern Observer

Code Smells	Debt (Hour, minutes)	Debt Ratio	Rating	Effort to Reach A
11	59 min	0,10%	A	0

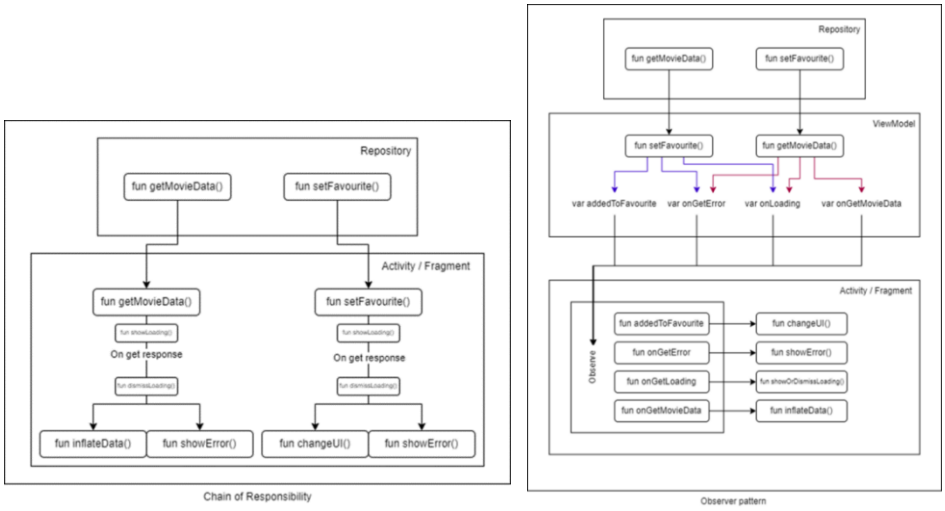
Hasil pengukuran *maintainability* yang dapat dilihat pada Tabel 1 dan Tabel 2 menunjukkan bahwa variabel Debt dan Debt ratio menjadi indikator utama yang membedakan aplikasi yang menggunakan desain pattern Observer dengan yang tidak menggunakan desain tersebut. Hasil analisis pada kedua versi aplikasi menunjukkan perbedaan yang cukup mencolok. Pada aplikasi yang mengadopsi desain pattern *Chain of Responsibility*, tercatat bahwa Debt mencapai 5 jam 37 menit dengan Debt ratio sebesar 0.3%. Hal ini menunjukkan bahwa usaha yang dibutuhkan untuk memperbaiki maupun menghilangkan *code smell* atau kode yang berpeluang menimbulkan masalah pada aplikasi ini cukup signifikan, dan persentase biaya usaha perbaikan juga relatif tinggi.

Sementara itu, pada aplikasi yang menggunakan desain pattern Observer, Debt hanya sekitar 59 menit dengan Debt ratio sekitar 0.1%. Hasil ini menunjukkan bahwa aplikasi dengan desain Observer memiliki tingkat maintainability yang lebih baik dengan keterpautan debt sebesar 4 jam 38 menit dan debt ratio sebasar 0.2%, dengan usaha perbaikan yang lebih rendah dan rasio biaya yang lebih efisien. Dengan demikian, dapat disimpulkan bahwa desain pattern Observer mungkin menjadi pilihan yang lebih baik untuk aplikasi yang mengutamakan maintainability yang tinggi dan mengurangi *code smell* atau kode yang berpeluang menimbulkan masalah.

Sebelum diperbaiki, aplikasi menggunakan pola distribusi data yang mengakibatkan *boilerplate code* yang banyak dan kompleksitas tinggi, karena data dari API langsung diteruskan oleh repository ke antarmuka pengguna (UI). Setelah perbaikan dilakukan, penyebaran data ditingkatkan dengan menyimpan data dari API dalam 'LiveData' di 'ViewModel'. Data ini kemudian diobservasi oleh *subscriber* di fragment, menciptakan struktur yang lebih terorganisir dan modular. Perbaikan ini mengurangi kebutuhan akan boilerplate code dalam UI, menyebabkan peningkatan dalam *maintainability* dan kualitas kode secara keseluruhan pada aplikasi.

3.5. Evaluasi Hasil Penelitian

Perubahan signifikan dalam *re-engineering* ini dapat dilihat melalui perbandingan diagram sebelum dan setelah perbaikan seperti yang ditunjukkan pada Gambar 5 berikut.



Gambar 5 Diagram Area Refaktor Pada Aplikasi Katafilm

Dari Gambar 5 di atas terlihat bahwa pada desain pattern *Observer* terdapat satu 'LiveData' yang diperuntukkan bagi satu tugas, namun 'LiveData' tersebut dapat menerima perubahan data berkali-kali dari berbagai fungsi yang memerlukan tempat untuk menampung data yang akan dipublikasikan. Hal ini memungkinkan penanganan data yang dinamis tanpa perlu menulis ulang kode yang sama. Di sisi lain, dalam desain pattern CoR, setiap kali sebuah fungsi menghasilkan data, maka diperlukan penulisan kode yang berulang sesuai dengan kebutuhan fungsional dari fungsi tersebut. Hal ini bisa mengakibatkan duplikasi kode yang tidak efisien dan kompleksitas yang tinggi dalam manajemen data. Dari perbandingan ini, desain pattern Observer cenderung memberikan struktur yang lebih modular dan efisien dalam penanganan data yang berubah secara dinamis dalam aplikasi.

Dari analisis maintainability yang dilakukan, aplikasi dengan desain pattern Observer ternyata menunjukkan nilai debt dan debt ratio yang lebih rendah dibandingkan dengan desain pattern CoR, menandakan perbaikan kode yang lebih efisien dan *maintainability* yang lebih baik. Perbedaan ini dapat diatribusikan pada klarifikasi peran yang jelas antara *publisher* (ViewModel) dan *subscriber* (fragment atau activity) dalam desain pattern *Observer*. Kelebihan ini mengurangi instruksi berulang pada *publisher*, menciptakan efisiensi jalur program dan nilai debt yang lebih rendah. Dengan demikian, desain pattern *Observer* menonjol sebagai pilihan yang mendukung *maintainability* yang lebih baik dalam aplikasi.

4. Kesimpulan

Berdasarkan langkah-langkah penelitian yang dilakukan pada aplikasi KataFilm, desain pattern *Observer* berhasil mengatasi permasalahan yang muncul pada implementasi sebelumnya yaitu desain pattern *Chain of Responsibility* (CoR). Penggunaan *Observer* memberikan fleksibilitas yang signifikan dalam menangani perubahan dinamis, memungkinkan penyesuaian respons sistem terhadap kebutuhan yang berkembang. Pemisahan antara subjek dan observer dalam model arsitektur *Observer* dapat memperjelas tanggung jawab masing-masing komponen, meningkatkan keterbacaan dan pemahaman kode, serta memudahkan pemeliharaan dan pengembangan lebih lanjut. Keleluasaan dalam penambahan dan penghapusan *observer* juga mempermudah manajemen komponen-komponen yang harus merespons perubahan. Model arsitektur *Observer* membantu meningkatkan kinerja dengan mengurangi ketergantungan dan overhead pada handler dalam rantai, menciptakan respons sistem yang lebih efisien. Secara keseluruhan, refaktor ke desain pattern *Observer* berhasil meningkatkan adaptabilitas dan fleksibilitas aplikasi KataFilm. Dengan demikian, implementasi desain pattern *Observer* terbukti sebagai langkah yang tepat untuk mengatasi permasalahan sebelumnya dan meningkatkan kualitas sistem secara menyeluruh.

Referensi

[1] H. Wang, H. Li, L. Li, Y Guo and G. Xu " Why are Android apps removed from Google Play?: a large-scale empirical study". ACM Digital Library, MSR '18: Proceedings of the 15th International Conference on Mining Software Repositories, 2018. Doi: https://doi.org/10.1145/3196398.3196412

[2] M. Ahasanuzzaman, S. Hassan, C. P. Bezemer, A. E Hassan. "A longitudinal study of popular ad libraries in the Google Play Store. Empir Software Eng 25, 824–858". 2020. Doi: https://doi.org/10.1007/s10664-019-09766-x

[3] D. Rimawi and S. Zein, "A Model Based Approach for Android Design Patterns Detection," *IEEE Explore, 3rd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT), 2019*. pp. 1–10. Doi: https://doi.org/10.1109/ISMSIT.2019.8932921

[4] P. Manh, "Chain of Responsibility pattern", *Github*, 2019. Dapat di akses di https://ducmanhphan.github.io/2019-02-27-Chain-of-Responsibility-pattern/.

[5] M. O. Onarcan and Y. Fu, "A Case Study on Design Patterns and Software Defects in Open Source Software", *Journal of Software Engineering and Applications,* 2018, vol.11 no.05. Doi: 10.4236/jsea.2018.115016

[6] D. Firmansyah. "Optimasi Maintainability Menggunakan Metode Clean Code Pada Sistem Informasi Akademik Sekolah," *Other Thesis, Universitas Komputer Indonesia*, 2020. URL: http://elibrary.unikom.ac.id/id/eprint/2763.

[7] P. W. Wirawan. "Model-View-Controller (MVC) Design Pattern Untuk Aplikasi Perangkat Bergerak Berbasis Java," *Teknik Informatika, Universitas Diponegoro*. 2010

[8] A. R. Fajri and S. Rani. "Penerapan Design Pattern MVVM dan Clean Architecture Pada Pengembangan Aplikasi Android (Studi Kasus: Aplikasi Agree)," *Automata, Journal Universitas Islam Indonesia*, vol. 3. No.2, 2022.

[9] A. F. Rahmawati and Y. A. Susetyo "Analisis Quality Code Menggunakan Sonarqube Dalam Suatu Aplikasi Berbasis Laravel", IT-EXPLORE, Vol. 2 No. 2 (2023): IT-Explore Juni 2023, https://doi.org/10.24246/itexplore.v2i2.2023.pp99-103

[10] V. Sarcar "", Observer Pattern. In: Java Design Patterns" Springer Link, 2022. pp 343–365. Doi: https://doi.org/10.1007/978-1-4842-7971-7_16