ISSN : 2614-0985 (Cetak)
ISSN : 2614-0977 (Online)
KOMPUTEK
url : http://studentjournal.umpo.ac.id/index.php/komputek

Implementasi *Builder Design Pattern* dalam Pembangunan Sistem Notifikasi yang Fleksibel pada Aplikasi Android

Aditya Daffa Syahputra*, Risma Auliya Salsabilla, Carudin

Fakultas Ilmu Komputer, Universitas Singaperbangsa Karawang

E-mail Korespondensi: 2210631170001@student.unsika.ac.id

History Artikel

Diterima : 5 Mei 2025 Disetujui : 10 September 2025 Dipublikasikan : 25 Oktober 2025

Abstract

The rapid advancement of software engineering in the Industry 4.0 era has made mobile applications, especially on Android, vital communication tools. Notification systems are a crucial element, but their diverse configurations often lead to inefficient and difficult-to-maintain code. Complex notifications with various parameters (e.g., icons, titles, content, priority, sounds, vibrations, images, and action buttons) are frequently built using lengthy constructors or long chains of setter methods. This approach reduces code readability and flexibility. This research aims to optimize the development of a flexible and manageable notification system in Android applications by implementing the Builder Design Pattern. Our research methodology uses an implementative case study approach and a prototype model. We built a simulated Android application in Kotlin, utilizing NotificationCompat.Builder to represent the Builder Design Pattern. The case study specifically focused on comparing conventional notification structures with those based on the Builder Design Pattern. Our stages included identifying and analyzing a case study for applying the design pattern to notifications, followed by creating an application prototype. The research findings show that the Builder Design Pattern significantly improves the readability, flexibility, and maintainability of notification code. Implementing this pattern allows for gradual and expressive notification configuration through method chaining (e.g., .setIcon().setTitle().setContent().build()), which transforms a complex process into an intuitive one. This can reduce code redundancy and simplify adding new features without altering the core structure. Notifications can be created with only the necessary parameters, which helps reduce complexity. As a result, the Builder Design Pattern is effective in simplifying the notification coding process, making it highly relevant for modern multi-channel applications and supporting a developer-friendly Android architecture.

Keywords: *Builder Design Pattern, Android Notification System, Code Readability, Kotlin, Mobile Application Development*

Abstrak

Pesatnya kemajuan rekayasa perangkat lunak di era Industri 4.0 mendorong aplikasi *mobile*, khususnya Android, sebagai media komunikasi vital. Sistem notifikasi menjadi elemen krusial, tetapi keragaman konfigurasinya sering mengakibatkan kode tidak efisien dan sulit dipelihara. Notifikasi kompleks dengan berbagai parameter (misalnya, ikon, judul, isi, prioritas, suara, getaran, gambar, dan tombol aksi) sering dibuat melalui konstruktor panjang atau metode *setter* berantai, yang menurunkan keterbacaan dan fleksibilitas kode. Penelitian ini bertujuan untuk mengoptimalkan pembangunan sistem notifikasi yang fleksibel dan mudah dikelola pada aplikasi Android dengan mengimplementasikan *Builder Design Pattern*. Metode penelitian menggunakan pendekatan studi kasus implementatif dan model *prototype*. Aplikasi simulasi Android dibangun dengan Kotlin, memanfaatkan NotificationCompat.Builder sebagai representasi pola desain tersebut. Studi kasus berfokus pada perbandingan antara struktur notifikasi konvensional dan berbasis *Builder Design Pattern*. Tahapan meliputi pencarian dan analisis studi kasus untuk penerapan pola desain pada notifikasi, dilanjutkan dengan pembuatan *prototype* aplikasi. Hasil penelitian menunjukkan bahwa *Builder Design Pattern* secara signifikan meningkatkan keterbacaan, fleksibilitas, dan kemudahan pemeliharaan kode notifikasi. Penerapan pola ini memungkinkan konfigurasi notifikasi secara bertahap dan ekspresif melalui *method chaining* (misalnya, .setIcon().setTitle().setContent().build()), yang mengubah proses kompleks menjadi intuitif. Hal tersebut dapat mengurangi redundansi kode dan mempermudah penambahan fitur tanpa mengubah struktur inti. Notifikasi dapat dibuat hanya dengan parameter yang dibutuhkan untuk mengurangi kerumitan. Hasilnya, *Builder Design Pattern* efektif untuk menyederhanakan proses pengkodean notifikasi yang menjadikannya relevan pada kebutuhan aplikasi *multi-channel* modern dan mendukung arsitektur Android yang *developer-friendly*.

Kata Kunci: *Builder Design Pattern, Sistem Notifikasi Android, Keterbacaan Kode, Kotlin, Pengembangan Aplikasi Mobile*

How to Cite: Aditya Daffa Syahputra (2025). Implementasi Builder Design Pattern dalam Pembangunan Sistem Notifikasi yang Fleksibel pada Aplikasi Android. KOMPUTEK : Jurnal Universitas Muhammadiyah Ponorogo, Vol 9(2): Halaman 67-75

ISSN 2614-0985 (Print)
ISSN 2614-0977 (Online)

PENDAHULUAN

Pada era industri saat ini, kemajuan pesat di bidang rekayasa perangkat lunak telah menjadikan pengembangan aplikasi sebagai aspek yang semakin krusial (Anastasia & Paputungan 2022). Seiring dengan dinamisnya teknologi tersebut, jangkauan serta keberagaman penggunaan aplikasi juga meluas secara signifikan dan kini merambah hampir kepada setiap individu, dari usia muda hingga dewasa, yang memiliki perangkat elektronik, seperti *smartphone* maupun komputer (Ivanka & Firdaus 2024). Dalam konteks aplikasi *mobile*, khususnya Android, sistem notifikasi menempati posisi vital dalam pengalaman pengguna. Fungsi utamanya adalah sebagai jembatan komunikasi yang efektif antara aplikasi dan penggunanya. Menurut (Andri et al. 2020), secara etimologis, "notifikasi" berasal dari bahasa Inggris "notification", yang berarti pengingat atau pemberitahuan melalui suatu media. Hal tersebut serupa dengan layanan *push notification* yang umum untuk mengirim pesan singkat langsung ke *smartphone* Android (Al Maliki et al., 2021).

Bagi (Rusdiana & Setiawan 2018), Android merupakan sistem operasi *open-source* berbasis Linux yang dominan pada *smartphone* dan tablet layar sentuh. Berdasarkan pernyataan dari (Kusuma 2018), Android menawarkan fleksibilitas luas bagi pengembang untuk melakukan inovasi. Dalam lingkungan ini, notifikasi yang efektif tidak hanya berperan besar dalam meningkatkan keterlibatan pengguna, tetapi juga memastikan informasi penting tersampaikan tepat waktu. Namun, di samping fungsionalitas, pengembangan aplikasi juga menuntut perhatian pada kualitas kode, antara lain harus mudah dibaca (*human readable*), cepat dalam pembangunan, dan mudah dikembangkan, baik secara individual maupun dalam tim (Dürschmid 2016). Kebutuhan terkait berbagai jenis notifikasi, seperti notifikasi teks sederhana, notifikasi bergambar, notifikasi dengan aksi, dan notifikasi progres, yang menggunakan konfigurasi beragam, sering kali menyebabkan kode menjadi berantakan, sulit dipahami, dan rentan terhadap kesalahan.

Oleh karena itu, salah satu faktor krusial dalam pengembangan aplikasi adalah memastikan kinerja optimal, yang meliputi kecepatan eksekusi, efisiensi penggunaan memori, dan pemanfaatan CPU yang efektif. Untuk mencapai hal tersebut, penerapan *design pattern* dalam pemrograman menjadi solusi yang sangat relevan (Putraadinatha et al. 2021). *Design pattern* sendiri merupakan solusi umum dan terbukti dapat mengatasi masalah desain secara berulang yang kerap muncul selama pengembangan aplikasi (Marchesi et al. 2021). Pola-pola ini dapat diibaratkan sebagai *blueprint* siap pakai yang dapat disesuaikan untuk menyelesaikan persoalan desain berulang dalam pengkodean (Drozdov et al. 2020). Secara umum, *design pattern* dikategorikan menjadi tiga kelompok berdasarkan sifatnya, yaitu *Creational* yang berfokus pada mekanisme pembuatan objek, *Behavioral* yang berkonsentrasi pada komunikasi antar objek, serta *Structural* yang berkaitan dengan komposisi kelas dan objek (RefactoringGuru).

Di antara beberapa kelompok *design pattern* tersebut, *Builder Design Pattern*, yang merupakan salah satu jenis

dari *Creational*, menonjol sebagai pola yang sering digunakan, khususnya untuk pembangunan objek yang kompleks. Pola tersebut memecah proses pembuatan objek yang rumit menjadi bagian-bagian yang lebih sederhana sehingga dapat menyederhanakan konstruksi dan meningkatkan fleksibilitas dalam memodifikasi komponen objek tersebut (Abdukadirova 2021). *Builder Design Pattern* memungkinkan pembuatan objek dengan berbagai variasi secara mudah karena aturan konstruksi yang melibatkan fungsi sebagai argumen dievaluasi selama tahap pembangunan (Girard et al. 2019). Dengan menyediakan antarmuka yang lebih sederhana, pola desain ini mempermudah proses pembuatan objek kompleks melalui pemanggilan metode untuk mengisi nilai secara individual (Pivarski et al. 2020). Secara esensial, *Builder Design Pattern* umumnya terdiri dari dua komponen utama, yaitu *builder* untuk membuat objek dan *director* untuk mengarahkan *builder* dalam proses konstruksi yang benar. Prinsip dasarnya adalah memisahkan proses pembuatan objek dari kelas utama yang secara signifikan dapat meningkatkan fleksibilitas dan mempermudah pemeliharaan kode (Khairunisa et al. 2024).

Dalam konteks pengembangan sistem notifikasi yang fleksibel menggunakan *Builder Design Pattern*, Kotlin menjadi pilihan bahasa pemrograman yang relevan. Kotlin merupakan bahasa pemrograman modern yang dirancang khusus untuk pengembangan aplikasi berbasis Android. Dengan sintaks yang ringkas, keamanan terhadap *null-pointer exception*, dan integrasi lintas platform secara penuh menggunakan Java, Kotlin mampu meningkatkan produktivitas pengembang. Sejak mendapat dukungan resmi dari Google pada tahun 2017, Kotlin semakin populer digunakan untuk proyek skala kecil maupun besar (Aulia & Cuhenda 2025). Meskipun Java, yang dikembangkan oleh Sun Microsystems pada 1995, populer dengan prinsip "Write Once, Run Anywhere (WORA)", Kotlin, yang dikembangkan JetBrains dan diumumkan pada 2011, dirancang untuk mengatasi beberapa kekurangannya sembari tetap berjalan di Java Virtual Machine (JVM) (Temaluru et al. 2025).

Berdasarkan latar belakang masalah dan tinjauan pustaka yang telah dilakukan, penelitian ini bertujuan untuk membandingkan implementasi sistem notifikasi Android menggunakan *Builder Design Pattern* dengan pendekatan konvensional. Konfigurasi notifikasi yang kompleks, seperti notifikasi dengan teks sederhana, notifikasi yang menyertakan gambar, dan notifikasi dengan tombol aksi interaktif, sering kali membutuhkan banyak parameter yang berbeda. Dalam struktur konvensional, hal tersebut dapat diartikan bahwa penggunaan konstruktor dengan banyak argumen (*telescoping constructor*) atau serangkaian panjang pemanggilan metode *setter* yang membuat kode tidak intuitif dan sulit untuk dipahami. Sebaliknya, penerapan *Builder Design Pattern* memungkinkan konfigurasi notifikasi secara bertahap dan ekspresif yang dapat membuat kode menjadi lebih mudah untuk dibaca, baik bagi pengembang itu sendiri maupun tim. Hasil studi ini diharapkan menjadi panduan praktis bagi pengembang lain yang menghadapi tantangan serupa dalam

membangun sistem notifikasi yang fleksibel dan mudah dikelola.

METODE PENELITIAN

Penelitian ini menggunakan metode pendekatan studi kasus implementatif, yang difokuskan pada penerapan *Builder Design Pattern* dalam pembuatan sistem notifikasi pada aplikasi Android. Pendekatan tersebut dilakukan agar fokus dari pembahasan dapat merujuk pada penerapan *Builder Design Pattern* dan gambaran baris kode yang menerapkan pola desain pada aplikasi tersebut. Studi ini dilakukan dengan membangun aplikasi sederhana yang memiliki dua tombol pemicu notifikasi dengan konfigurasi konten berbeda, tetapi menggunakan struktur konfigurasi notifikasi yang serupa. Implementasi dilakukan menggunakan bahasa pemrograman Kotlin dengan memanfaatkan 'class NotificationCompat.Builder' secara langsung yang merepresentasikan *Builder Design Pattern*.

Metode pengembangan aplikasi mengikuti model *prototype*, di mana versi awal dari sistem notifikasi dibuat, diuji coba, dan diperbaiki secara iteratif hingga mencapai bentuk yang optimal sebagai demonstrasi dari pola desain yang diteliti. Terdapat beberapa tahapan dalam penelitian yang dilakukan, yaitu mencari studi kasus, menganalisis studi kasus, dan pembuatan *prototype* aplikasi menggunakan *Builder Design Pattern.*

Tahap pertama yang dilakukan adalah pencarian studi kasus yang berkaitan dengan penerapan *Builder Design Pattern* dalam pembangunan sebuah fitur pada aplikasi Android. Tahap selanjutnya, yaitu dilakukan analisis bagaimana *Builder Design Pattern* dapat diterapkan pada fitur aplikasi yang dipilih dan mencari informasi bagaimana proses penerapan pola tersebut pada aplikasi Android. Tahap terakhir ialah pembuatan aplikasi yang dirancang berdasarkan hasil analisis yang telah dilakukan sebelumnya dan berfokus pada proses pengembangan sistem notifikasi yang menerapkan *Builder Design Pattern*.

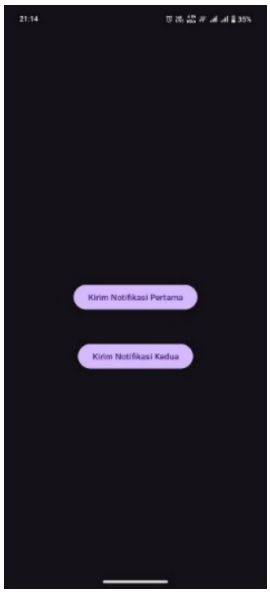
Secara keseluruhan, metode penelitian ini menggabungkan pendekatan studi kasus implementatif dan model pengembangan *prototype*. Setiap tahapan, mulai dari pencarian dan analisis studi kasus hingga pembangunan aplikasi Android, dilakukan identifikasi dan evaluasi bagaimana *Builder Design Pattern* dapat diterapkan secara nyata dalam pengembangan sistem notifikasi di aplikasi Android. Hasil dari proses ini diharapkan mampu memberikan gambaran konkret tentang efektivitas dan fleksibilitas pola desain tersebut dalam konteks pengembangan aplikasi berbasis *mobile*.

HASIL DAN PEMBAHASAN

Studi kasus yang diambil dalam penelitian ini adalah pembangunan sistem notifikasi pada Android yang kompleks, mencakup beberapa atribut dasar, seperti ikon, judul, isi pesan, prioritas, serta fitur tambahan seperti suara, getaran, notifikasi bergambar, dan tombol aksi. Dalam kondisi tanpa *Builder Design Pattern*, konfigurasi notifikasi semacam ini umumnya dilakukan melalui konstruktor dengan banyak parameter (*telescoping constructor*) yang sulit dibaca dan rentan terhadap kesalahan, serta serangkaian panjang pemanggilan metode *setter* yang menghasilkan kode tersebar dan tidak kohesif. Kedua pendekatan konvensional ini mengakibatkan kode menjadi berantakan, sulit diinterpretasikan secara sekilas, dan rumit untuk dimodifikasi.

Kemudian, dilakukan analisis studi kasus berdasarkan penelitian terdahulu yang menghasilkan temuan bahwa implementasi *Builder Design Pattern* secara signifikan dapat meningkatkan fleksibilitas, keterbacaan kode, dan kemudahan dalam pemeliharaan. Pada penelitian ini, aplikasi dirancang sebagai media simulasi untuk menampilkan notifikasi dengan berbagai konfigurasi berbeda. Hal tersebut dicapai melalui pemanfaatan NotificationCompat.Builder sebagai representasi dari *Builder Design Pattern* yang beroperasi secara manual, lalu dibandingkan dengan struktur konvensional. Analisis menunjukkan bahwa pola ini memungkinkan konfigurasi notifikasi secara bertahap dan ekspresif melalui metode berantai. Contohnya adalah .setIcon().setTitle().setContent().build(), yang mengubah proses pembangunan notifikasi kompleks menjadi alur yang sangat mudah dibaca, layaknya membaca kalimat secara eksplisit yang menjelaskan setiap atribut yang diatur. Pola ini juga memberikan fleksibilitas tinggi dalam konfigurasi, memungkinkan pembuatan notifikasi dengan berbagai kombinasi fitur tanpa perlu menulis ulang konstruktor atau mengelola *setter* yang panjang.

Selanjutnya, *prototype* aplikasi dibuat menggunakan bahasa pemrograman Kotlin dan memiliki tampilan utama dengan dua tombol sebagai *trigger* dari sistem notifikasi yang divisualisasikan pada Gambar 1.



Gambar 1. Tampilan Aplikasi

Aplikasi ini dirancang sebagai media simulasi untuk menampilkan notifikasi dengan konfigurasi berbeda dan melalui pemanfaatan struktur ‘NotificationCompat.Builder’ sebagai representasi dari *Builder Design Pattern* dengan struktur manual yang dapat dilihat lebih jelas pada Gambar 2.

```
private fun sendNotification(title: String, message: String) {
    val notificationManager = getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
    val builder = NotificationCompat.Builder( context: this, CHANNEL_ID_1)
        .setContentTitle(title)
        .setContentText(message)
        .setSmallIcon(R.drawable.ic_notification)
        .setPriority(NotificationCompat.PRIORITY_DEFAULT)
        .setSubText(getString(R.string.notification_channel_name1))
```

Gambar 2. Penerapan Kode *Builder Design Pattern* pada Fungsi SendNotification() dengan 5 Parameter

Baris kode di atas merupakan tahapan dalam pembuatan notifikasi yang mengandung konsep *Builder Design Pattern.* Terdapat beberapa kode yang digunakan sebagai parameter ‘NotificationCompat.Builder’ dalam membangun notifikasi dengan masing-masing fungsi yang dapat dilihat pada Tabel 1, di antaranya adalah:

Tabel 1. Fungsi dari Parameter ‘NotificationCompat.Builder’

.setContentTitle(title)	Digunakan untuk mengatur isi utama teks notifikasi yang akan ditampilkan di bawah judul (*title*).

	Parameter ini adalah informasi inti dari notifikasi yang ingin disampaikan kepada pengguna.
.setContentText(message) .setSmallIcon(R.drawable.*ic_notification*)	Digunakan untuk menentukan ikon kecil yang muncul di bilah status (*status bar*) dan di sisi kiri notifikasi. Parameter ini adalah komponen visual yang wajib agar notifikasi valid.
.setPriority(NotificationCompat.*PRIORITY_DEFAULT*)	Digunakan untuk mengatur tingkat prioritas notifikasi. Nilai 'PRIORITY_DEFAULT' membuat notifikasi muncul normal tanpa mengganggu aktivitas pengguna secara berlebihan.
.setSubText(getString(R.string.*notification_channel_name1*))	Digunakan untuk menambahkan teks tambahan (*subtext*) di bawah isi utama notifikasi. Teks tersebut biasanya berupa kategori, sumber atau konteks tambahan.

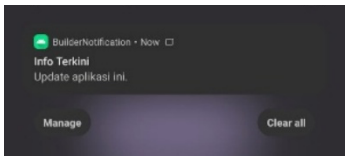
Kode di atas merupakan salah satu penerapan lengkap untuk membangun sebuah sistem notifikasi beserta parameternya menggunakan prinsip *Builder Design Pattern*. Dengan pola desain tersebut juga memungkinkan pembuatan notifikasi hanya dengan parameter yang dibutuhkan tanpa menggunakan keseluruhan parameter yang ada dan ditunjukkan pada Gambar 3.

```
val builder = NotificationCompat.Builder( context: this, CHANNEL_ID_2)
    .setContentTitle(title)
    .setContentText(message)
    .setSmallIcon(R.drawable.ic_notification)
    .setPriority(NotificationCompat.PRIORITY_DEFAULT)
```

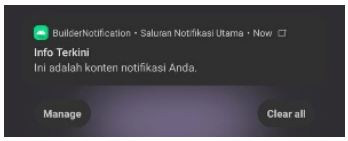
Gambar 3. Penerapan Kode *Builder Design Pattern* pada Fungsi SendNotification2() dengan 4 Parameter

Beberapa parameter dalam proses pembuatan notifikasi dirancang agar dapat disesuaikan dengan kebutuhan spesifik aplikasi. Hal tersebut memungkinkan pengembang untuk menciptakan pengalaman notifikasi yang relevan dan fungsional bagi pengguna.

Berikut adalah perbedaan dari tampilan notifikasi dengan 5 parameter pada Gambar 4 dan 4 parameter pada Gambar 5.



Gambar 4. Hasil Fungsi Notifikasi dengan 5 Parameter



Gambar 5. Hasil Fungsi Notifikasi dengan 4 Parameter

Tanpa adanya penerapan *Builder Design Pattern*, penulisan kode akan lebih rumit dan kurang efisien dalam hal keterbacaan kode yang dapat dilihat pada Gambar 6.

```
val notification = Notification()
notification.icon = R.drawable.ic_notification
notification.tickerText = title
```

Gambar 6. Contoh Pembuatan Variabel Notifikasi Tanpa Menggunakan *Builder Design Pattern*

Dari hasil implementasi, penggunaan 'NotificationCompat.Builder' menunjukkan penerapan *Builder Design Pattern* secara nyata, di mana konfigurasi notifikasi disusun secara fleksibel melalui *method chaining*. Pendekatan ini jauh

lebih efisien dibandingkan dengan metode lama yang bersifat prosedural, terlebih karena metode seperti setLatestEventInfo() yang tidak dipergunakan lagi. Penerapan *Builder Design Pattern* juga tidak hanya menyederhanakan penulisan kode, tetapi meningkatkan keterbacaan, skalabilitas, dan kemudahan modifikasi struktur notifikasi. Proses pembangunan sistem notifikasi yang sebelumnya rentan terhadap kesalahan dan sulit dikelola, kini menjadi lebih terstruktur, intuitif, dan *developer-friendly*.

Meskipun *Builder Design Pattern* telah dikenal luas dalam pengembangan perangkat lunak, penelitian ini menghadirkan keterbaruan dalam konteks penerapannya pada sistem notifikasi Android modern menggunakan Kotlin dan 'NotificationCompat.Builder'. Fokus pada penelitian ini adalah perbandingan pendekatan struktural notifikasi antara teknik prosedural dan berbasis *Builder Design Pattern*, khususnya dalam skenario *multi-channel notification* yang memberikan kontribusi praktis dalam membangun aplikasi modular dan *maintainable*. Penelitian ini juga memperkuat relevansi pola desain tersebut terhadap kebutuhan desain antarmuka sistem notifikasi yang dinamis dan kontekstual pada Android terbaru.

KESIMPULAN

Berdasarkan hasil dan pembahasan yang telah dijelaskan menunjukkan bahwa penerapan *Builder Design Pattern* dalam sistem notifikasi pada Android tidak hanya menyederhanakan proses pengkodean, tetapi relevan terhadap kebutuhan aplikasi masa kini yang mendukung *multi-channel* dan fleksibilitas konfigurasi. Keterbaruan dari penelitian ini terletak pada demonstrasi implementatif dan analisis penerapan pola desain tersebut ke dalam lingkungan Kotlin dan API Android terbaru, yang belum banyak dieksplorasi dalam literatur ilmiah lokal. Pendekatan ini dapat menjadi acuan praktis bagi pengembang dalam membangun notifikasi yang efisien dan mudah dipelihara.

Penerapan *Builder Design Pattern* dalam pengembangan sistem notifikasi pada Android memberikan solusi yang efektif dalam membangun konfigurasi notifikasi yang kompleks dengan cara terstruktur dan mudah dipahami. Dengan memanfaatkan 'NotificationCompat.Builder', pengembang dapat dengan mudah menyesuaikan parameter notifikasi sesuai kebutuhan tanpa membebani kode menggunakan konfigurasi yang tidak diperlukan. Penelitian ini membuktikan bahwa *Builder Design Pattern* sangat relevan untuk meningkatkan kualitas desain perangkat lunak pada fitur sederhana, serta menjadi pendekatan yang sesuai dengan perkembangan arsitektur Android saat ini.

DAFTAR PUSTAKA

Abdukadirova, M. A. (2021). The Role of Builder and Building in the Development of the Country is Invaluable. *The American Journal of Interdisciplinary Innovations and Research*, *3*(05), 81-84.

Al Maliki, M. A., Sya'roni, W., & Sudriyanto, S. (2021). Sistem Informasi Notifikasi Kegiatan Masjid Mujahidin Menggunakan Android. *JATISI (Jurnal Teknik Informatika dan Sistem Informasi)*, *8*(4), 1832-1843.

Anastasia, F. D., & Paputungan, I. V. (2022). Implementasi Bloc Pattern Pada Pengembangan Frontend Fitur Top Up Linkaja Aplikasi M-Banking Agen46 Dengan Teknologi Flutter (Studi Kasus: PT. Bank Negara Indonesia

Tbk). *Jurnal Sains, Nalar, Dan Aplikasi Teknologi Informasi*, *2*(1), 1-11.

Andri, R., Saputri, N. A. O., & Akbar, M. (2020). Sistem Notifikasi Tugas Akhir Universitas Bina Darma Berbasis Mobile. *Sistemasi: Jurnal Sistem Informasi*, *9*(1), 155-165.

Aulia, R., & Cuhenda, C. (2025). Perancangan Aplikasi Customer Relationship Management (CRM) Untuk PT. Wahana Satu Propertindo. *Jurnal Informatika dan Teknologi Pendidikan*, *5*(1).

Drozdov, D., Atmojo, U. D., Pang, C., Patil, S., Ali, M. I., Tenhunen, A., ... & Vyatkin, V. (2020). Utilizing software design patterns in product-driven manufacturing system: A case study. In *Service Oriented, Holonic and Multi-agent Manufacturing Systems for Industry of the Future: Proceedings of SOHOMA 2019 9* (pp. 301-312). Springer International Publishing.

Dürschmid, T. (2016). Design pattern builder: a concept for refinable reusable design pattern libraries. In *Companion Proceedings of the 2016 ACM SIGPLAN International Conference on Systems, Programming, Languages and Applications: Software for Humanity* (pp. 45-46).

Girard, M., Ehlen, A., Shakya, A., Bereau, T., & de la Cruz, M. O. (2019). Hoobas: A highly object-oriented builder for molecular dynamics. *Computational Materials Science*, *167*, 25-33.

Ivanka, A. A., & Firdaus, M. E. B. (2024). Fico Words: Perancangan Game Antarmuka Tebak Kata Interaktif Berbasis Android untuk Anak Usia Dini Mengenal Huruf dan Kata. *KOMPUTEK*, *8*(1), 51-60.

Khairunisa, N., Rishwan, R. M., Saputro, R. A., & Carudin, C. (2024). Pengaruh Builder Design Pattern Pada Performa Aplikasi Dengan Menggunakan Bahasa Pemrograman Golang. *Majalah Ilmiah UNIKOM*, *22*(1), 3-8.

Kusuma, W. A. (2018). Sistem Informasi Geografis Pemetaan Lokasi Bird Contest Kota Malang Berbasis Android. *SISTEMASI: Jurnal Sistem Informasi*, *7*(3), 212-219.

Marchesi, L., Marchesi, M., Destefanis, G., Barabino, G., & Tigano, D. (2020). Design patterns for gas optimization in ethereum. In *2020 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE)* (pp. 9-15). IEEE.

Pivarski, J., Elmer, P., & Lange, D. (2020). Awkward arrays in python, c++, and numba. In *EPJ Web of Conferences* (Vol. 245, p. 05023). EDP Sciences.

Putraadinatha, I. G. B. V., Suwawi, D. D. J., & Puspitasari, S. Y. (2021). Pengaruh Design Pattern Terhadap Maintainability Aplikasi Mobile. *eProceedings of Engineering*, *8*(5).

RefactoringGuru. Design Pattern. Tersedia di: https:// refactoring.guru/design-patterns. Html. Diakses pada 05 Juni 2025.

Rusdiana, L., & Setiawan, H. (2018). Perancangan Aplikasi Monitoring Kesehatan Ibu Hamil Berbasis Mobile Android. *Sistemasi: Jurnal Sistem Informasi*, *7*(3), 197-203.

Temaluru, M. A. G., Rivandy, P., & Pedo, J. (2025). Perancangan Aplikasi Pendaftaran Siswa Baru Berbasis Android Untuk Meningkatkan Efisiensi Proses Pendaftaran. *Increate-Inovasi dan Kreasi Teknologi*, *11*(1).