AN-NURIYAH
JOURNAL OF ISLAMIC TECHNOLOGY AND INFORMATICS EDUCATION
OPEN ACCESS ISSN 3123-6170
AN-NURIYAH
Jurnal of Islamic Technology and Informatics Education
Vol. 2 No. 01 Tahun 2026 Page [128-141]

Pengaruh Penerapan Prinsip SOLID Dan Design Patterns Terhadap Maintainability Kode Sumber Perangkat Lunak

[1]Abdul Jabbar [2]Abdullah
[1,2]Politeknik IDN Bogor, Indonesia
Email: akujaber12@gmail.com

Abstrak

Maintainability merupakan salah satu atribut kualitas perangkat lunak yang sangat penting, terutama dalam pengembangan sistem berskala menengah hingga besar. Kode sumber yang sulit dipelihara akan meningkatkan biaya pengembangan, memperlambat proses perbaikan, serta menyulitkan pengembangan fitur baru. Permasalahan ini umumnya disebabkan oleh struktur kode yang kompleks, tingkat ketergantungan antar modul yang tinggi, serta desain perangkat lunak yang tidak terorganisasi dengan baik. Penelitian ini bertujuan untuk menganalisis pengaruh penerapan prinsip SOLID dan design patterns terhadap maintainability kode sumber perangkat lunak. Metode penelitian yang digunakan adalah pendekatan kualitatif deskriptif dengan studi kasus pada kode sumber perangkat lunak yang mengalami proses refactoring. Analisis dilakukan dengan membandingkan kondisi kode sebelum dan sesudah penerapan prinsip SOLID serta design patterns seperti Factory dan Strategy, dengan fokus pada aspek keterbacaan, modularitas, fleksibilitas, dan kemudahan pemeliharaan. Hasil penelitian menunjukkan bahwa penerapan prinsip SOLID mampu menurunkan kompleksitas desain dan meningkatkan pemisahan tanggung jawab kelas, sementara design patterns berkontribusi dalam meningkatkan fleksibilitas dan kemudahan pengembangan lanjutan. Meskipun terjadi peningkatan jumlah kelas dan abstraksi, kualitas desain kode secara keseluruhan mengalami peningkatan yang signifikan. Temuan ini menegaskan bahwa maintainability lebih ditentukan oleh kualitas desain daripada kuantitas baris kode. Dengan demikian, penelitian ini memberikan kontribusi teoretis dan praktis bahwa penerapan prinsip SOLID dan design patterns secara konsisten dapat menjadi solusi efektif dalam meningkatkan maintainability perangkat lunak.

Kata kunci: SOLID Principle, Design Patterns, Maintainability, Refactoring, Kualitas Perangkat Lunak.

Abstract

Maintainability is one of the most critical software quality attributes, particularly in the development of medium- to large-scale systems. Poorly maintainable source code can lead to increased development costs, slower bug fixing processes, and difficulties in extending system functionality. These issues are commonly caused by high code complexity, tight coupling between modules, and poorly structured software design. This study aims to analyze the impact of implementing SOLID principles and design patterns on software source code maintainability. The research adopts a descriptive qualitative approach using a case study of software source code that underwent a refactoring process. The analysis compares the conditions of the code before and after the implementation of SOLID principles and design patterns such as Factory and Strategy, focusing on readability, modularity, flexibility, and ease of maintenance. The results indicate that the application of SOLID principles significantly reduces design complexity and improves the separation of class responsibilities, while design patterns enhance system flexibility and support future feature development. Although the refactoring process increases the number of classes and abstractions, it leads to a substantial improvement in overall code quality and readability. These findings confirm that maintainability is determined not by the number of lines of code, but by the quality and organization of software design. Therefore, this study provides both theoretical and practical evidence that consistent application of SOLID principles and design patterns is an effective approach to improving software maintainability.

Keywords: *SOLID Principles, Design Patterns, Maintainability, Refactoring, Software Quality.*

PENDAHULUAN

Perkembangan teknologi informasi yang semakin pesat telah mendorong lahirnya berbagai perangkat lunak dengan tingkat kompleksitas yang tinggi. Perangkat lunak tidak lagi hanya berperan sebagai alat pendukung operasional, tetapi telah menjadi komponen strategis dalam hampir seluruh sektor, seperti bisnis, pendidikan, pemerintahan, dan industri. Kondisi ini menuntut perangkat lunak untuk tidak hanya memenuhi kebutuhan fungsional pengguna, tetapi juga memiliki kemampuan adaptasi yang baik terhadap perubahan, mudah dikembangkan, serta efisien dalam jangka panjang. Perubahan kebutuhan pengguna, evolusi teknologi, dan tuntutan peningkatan fitur menjadikan keberlanjutan perangkat lunak sebagai tantangan utama dalam proses pengembangannya (Saad, A. 2025).

Dalam siklus hidup pengembangan perangkat lunak, fase pemeliharaan (maintenance) memegang peranan yang sangat penting. Berbagai penelitian dalam bidang rekayasa perangkat lunak menunjukkan bahwa sebagian besar biaya dan waktu pengembangan justru dihabiskan pada tahap pemeliharaan dibandingkan dengan tahap perancangan dan implementasi awal (Shrestha, J. 2025). Aktivitas pemeliharaan mencakup perbaikan kesalahan, penyesuaian terhadap perubahan lingkungan, peningkatan kualitas, serta pengembangan fitur baru. Oleh karena itu, kualitas kode sumber menjadi faktor penentu utama dalam efektivitas proses pemeliharaan dan keberlanjutan suatu perangkat lunak (Azhizi, M. H., & Yaqin, M. A. 2024).

Salah satu atribut kualitas perangkat lunak yang sangat berpengaruh terhadap keberhasilan pemeliharaan adalah *maintainability*. Maintainability mengacu pada sejauh mana kode sumber dapat dipahami, dianalisis, dimodifikasi, diuji, dan dikembangkan kembali dengan usaha yang relatif minimal. Kode sumber yang memiliki tingkat maintainability tinggi memungkinkan pengembang untuk melakukan perubahan secara lebih cepat dan aman, baik oleh pengembang awal maupun oleh pengembang lain yang

terlibat di kemudian hari. Sebaliknya, kode dengan maintainability rendah cenderung sulit dipahami, memiliki struktur yang kompleks, serta meningkatkan risiko munculnya kesalahan baru ketika dilakukan perubahan (Rhamadhan, W. S., Daniyudin, J., El Sofya, Dkk., 2025).

Dalam praktik pengembangan perangkat lunak, permasalahan rendahnya maintainability sering kali disebabkan oleh fokus pengembangan yang lebih menitikberatkan pada pencapaian fungsi bisnis jangka pendek. Tekanan tenggat waktu, keterbatasan sumber daya, serta kurangnya perhatian terhadap kualitas desain mengakibatkan kode sumber dikembangkan tanpa struktur yang baik. Kode yang tidak memiliki pemisahan tanggung jawab yang jelas, memiliki ketergantungan yang tinggi antar modul, dan minim perencanaan desain akan menyulitkan proses pengujian, refactoring, serta pengembangan lanjutan. Kondisi ini pada akhirnya berdampak pada meningkatnya kompleksitas kode dan tingginya biaya pemeliharaan (Kevin, A., & Kurniawan, H. C. 2025).

Permasalahan tersebut menunjukkan pentingnya penerapan prinsip desain perangkat lunak yang mampu mengelola kompleksitas dan meningkatkan kualitas struktur kode sumber. Dalam disiplin rekayasa perangkat lunak, prinsip SOLID dan *design patterns* diperkenalkan sebagai pendekatan desain yang bertujuan untuk menghasilkan perangkat lunak yang modular, fleksibel, dan mudah dipelihara. Prinsip SOLID, yang terdiri dari *Single Responsibility Principle*, *Open/Closed Principle*, *Liskov Substitution Principle*, *Interface Segregation Principle*, dan *Dependency Inversion Principle*, menekankan pada pemisahan tanggung jawab, pengurangan ketergantungan antar komponen, serta peningkatan fleksibilitas desain. Sementara itu, *design patterns* menyediakan solusi desain yang telah teruji dan dapat digunakan kembali untuk menyelesaikan permasalahan umum dalam pengembangan perangkat lunak (Alhunaiti, S. A. B. A., & Khan, M. S. 2023).

Secara konseptual, penerapan prinsip SOLID dan *design patterns* diyakini mampu meningkatkan kualitas dan maintainability kode sumber. Namun, dalam praktiknya penerapan kedua konsep tersebut masih belum dilakukan secara konsisten oleh banyak pengembang. Selain itu, kajian empiris yang secara khusus mengukur pengaruh penerapan prinsip SOLID dan *design patterns* terhadap maintainability kode sumber menggunakan metrik perangkat lunak yang terukur masih relatif terbatas. Oleh karena itu, diperlukan penelitian yang mampu memberikan bukti empiris mengenai sejauh mana penerapan prinsip SOLID dan *design patterns* berpengaruh terhadap maintainability kode sumber perangkat lunak.

METODE PENELITIAN

Penelitian ini menggunakan pendekatan kuantitatif dengan desain eksperimen semu (*quasi-experimental*) untuk menganalisis pengaruh penerapan prinsip SOLID dan *design patterns* terhadap maintainability kode sumber perangkat lunak. Objek penelitian berupa kode sumber perangkat lunak yang dikembangkan dalam dua kondisi, yaitu sebelum dan sesudah penerapan prinsip SOLID dan *design patterns*. Pada tahap awal, kode dikembangkan atau dianalisis tanpa menerapkan prinsip desain tersebut. Selanjutnya, kode sumber dilakukan proses *refactoring* dengan menerapkan prinsip SOLID dan beberapa *design patterns* yang relevan, tanpa mengubah fungsi utama sistem. Pendekatan ini bertujuan untuk memastikan bahwa perbedaan maintainability yang dihasilkan benar-benar disebabkan oleh perubahan pada struktur dan desain kode (Hilalludi., 2024).

Pengukuran maintainability dilakukan menggunakan metrik perangkat lunak yang bersifat objektif dan terukur, meliputi *Maintainability Index*, *Cyclomatic Complexity*, *Coupling Between Objects*, dan *Lines of Code*. Data yang diperoleh dari kedua kondisi kode kemudian dianalisis secara deskriptif dan komparatif untuk mengidentifikasi perubahan tingkat maintainability. Hasil analisis digunakan untuk menilai sejauh mana penerapan prinsip SOLID dan

design patterns berkontribusi dalam menurunkan kompleksitas, mengurangi ketergantungan antar modul, serta meningkatkan kemudahan pemeliharaan kode sumber perangkat lunak.

HASIL DAN PEMBAHASAN

Kondisi Maintainability Kode Sumber Sebelum Penerapan Prinsip SOLID dan Design Patterns

Hasil analisis awal terhadap kode sumber perangkat lunak yang dikembangkan tanpa penerapan prinsip SOLID dan *design patterns* menunjukkan bahwa tingkat maintainability berada pada kondisi yang relatif rendah. Struktur kode sumber cenderung bersifat kompleks dan kurang terorganisasi, di mana satu kelas sering kali menangani lebih dari satu tanggung jawab fungsional. Fenomena ini mencerminkan pelanggaran terhadap *Single Responsibility Principle*, yang secara teoritis menyatakan bahwa setiap modul atau kelas seharusnya memiliki satu alasan untuk berubah. Ketidakterapan prinsip ini menyebabkan meningkatnya kompleksitas internal kelas dan menyulitkan pengembang dalam memahami peran serta alur logika program secara komprehensif (Wang, Z., Ling, Dkj,. 2025).

Tingginya kompleksitas kode juga tercermin dari logika program yang saling terkait erat antar modul. Menurut teori *software complexity*, kompleksitas yang tinggi akan berdampak langsung pada menurunnya keterbacaan (*readability*) dan kemudahan analisis kode. Kode yang kompleks membutuhkan usaha kognitif yang lebih besar bagi pengembang untuk memahami hubungan antar bagian sistem, sehingga proses analisis, debugging, dan modifikasi menjadi lebih lambat dan rentan terhadap kesalahan. Kondisi ini sejalan dengan pandangan bahwa kompleksitas merupakan salah satu faktor utama yang menurunkan kualitas maintainability

perangkat lunak (*Applying SOLID principles for the refactoring of legacy code* 2024).

Selain kompleksitas, tingkat ketergantungan (*coupling*) antar modul pada kode sumber sebelum penerapan prinsip desain yang baik juga tergolong tinggi. Modul-modul dalam sistem saling bergantung secara langsung pada implementasi konkret, bukan pada abstraksi. Berdasarkan teori *coupling and cohesion*, tingginya *coupling* akan mengurangi fleksibilitas sistem dan meningkatkan risiko terjadinya efek domino ketika dilakukan perubahan. Setiap modifikasi pada satu modul berpotensi memengaruhi modul lain yang bergantung padanya, sehingga memperbesar kemungkinan munculnya kesalahan baru dan meningkatkan biaya pemeliharaan.

Kondisi ini menunjukkan bahwa kode sumber belum dirancang untuk menghadapi perubahan secara adaptif. Padahal, menurut konsep *software evolution*, perubahan merupakan keniscayaan dalam siklus hidup perangkat lunak. Sistem yang tidak dirancang dengan mempertimbangkan perubahan akan mengalami penurunan kualitas secara progresif seiring bertambahnya fitur dan kebutuhan baru. Rendahnya fleksibilitas desain pada kode sumber sebelum penerapan prinsip SOLID dan *design patterns* menegaskan bahwa perangkat lunak berada pada kondisi yang rentan terhadap degradasi kualitas dalam jangka panjang (*Understanding the importance of design patterns in software development*., 2025).

Lebih lanjut, lemahnya pemisahan tanggung jawab dan tingginya ketergantungan antar komponen berdampak langsung pada aspek lain dari maintainability, seperti *modifiability* dan *testability*. Kode yang sulit dipisahkan secara modular akan menyulitkan proses pengujian unit, karena satu modul tidak dapat diuji secara independen tanpa melibatkan modul lain. Hal ini bertentangan dengan prinsip desain perangkat lunak yang

menekankan pentingnya modularitas sebagai dasar untuk membangun sistem yang mudah diuji dan dipelihara.

Temuan-temuan tersebut memperkuat permasalahan yang telah diuraikan pada latar belakang penelitian, yaitu bahwa rendahnya maintainability kode sumber bukan semata-mata disebabkan oleh ukuran atau jumlah baris kode, melainkan oleh kualitas desain dan struktur kode itu sendiri. Dengan demikian, kondisi kode sumber sebelum penerapan prinsip SOLID dan *design patterns* dapat dipandang sebagai representasi dari praktik pengembangan yang kurang memperhatikan prinsip desain, yang pada akhirnya berdampak negatif terhadap maintainability perangkat lunak secara keseluruhan.

Perubahan Struktur Kode Setelah Penerapan Prinsip SOLID dan Design Patterns

Hasil proses *refactoring* kode sumber dengan menerapkan prinsip SOLID dan *design patterns* menunjukkan adanya perubahan struktural yang signifikan terhadap arsitektur perangkat lunak. Salah satu perubahan paling menonjol terlihat pada penerapan *Single Responsibility Principle* (SRP), yang mendorong pemisahan fungsi dan tanggung jawab kelas secara lebih jelas. Setiap kelas dirancang untuk menangani satu tujuan spesifik, sehingga kompleksitas internal kelas dapat ditekan. Menurut teori desain berorientasi objek, pemisahan tanggung jawab yang baik akan meningkatkan *cohesion* dan mengurangi beban kognitif pengembang dalam memahami logika program. Kondisi ini secara langsung berdampak pada meningkatnya keterbacaan dan kemudahan analisis kode sumber (Dharmayanti, D., & Bachtiar, A. M. 2025).

Selain itu, penerapan *Dependency Inversion Principle* (DIP) dan *Interface Segregation Principle* (ISP) memberikan kontribusi penting dalam menurunkan tingkat ketergantungan antar modul. Modul tingkat tinggi tidak lagi bergantung secara langsung pada implementasi konkret modul tingkat

rendah, melainkan bergantung pada abstraksi berupa antarmuka. Pendekatan ini sejalan dengan teori *loose coupling* dalam rekayasa perangkat lunak, yang menekankan bahwa sistem dengan tingkat ketergantungan rendah akan lebih mudah dipelihara dan dikembangkan. Dengan berkurangnya ketergantungan langsung antar komponen, perubahan pada satu modul tidak lagi menimbulkan dampak signifikan terhadap modul lain, sehingga risiko kesalahan akibat perubahan dapat diminimalkan.

Penerapan *Interface Segregation Principle* juga berperan dalam meningkatkan kualitas desain dengan memastikan bahwa setiap antarmuka hanya memuat metode yang benar-benar dibutuhkan oleh klien. Hal ini mencegah terjadinya *fat interface* yang memaksa kelas untuk mengimplementasikan metode yang tidak relevan. Berdasarkan teori desain modular, antarmuka yang spesifik dan terfokus akan meningkatkan fleksibilitas sistem serta memudahkan proses pengujian dan pengembangan lanjutan. Dengan demikian, kode sumber menjadi lebih adaptif terhadap perubahan kebutuhan tanpa harus melakukan modifikasi besar pada struktur yang sudah ada (Rochimah, S., Akbar, R. J., & Langsari, K. 2025).

Penggunaan *design patterns* seperti *Factory* dan *Strategy* semakin memperkuat fleksibilitas dan keteraturan struktur kode sumber. *Factory Pattern* membantu memisahkan proses pembuatan objek dari penggunaan objek itu sendiri, sehingga mendukung prinsip *Open/Closed Principle* dengan memungkinkan penambahan jenis objek baru tanpa memodifikasi kode yang sudah ada. Sementara itu, *Strategy Pattern* memungkinkan variasi perilaku sistem diatur melalui komposisi, bukan pewarisan, sehingga mempermudah penggantian atau penambahan algoritma tanpa mengganggu struktur utama sistem (Gamma, E., Helm, R., Johnson, R., & Vlissides, J. 2021). Menurut teori *reusable design*, pola-pola desain ini membantu menghindari solusi ad-hoc dan mendorong penggunaan struktur desain yang telah teruji dan mudah dipahami oleh pengembang lain (Ramachandrappa, N. C. 2024).

Secara keseluruhan, penerapan prinsip SOLID dan *design patterns* menghasilkan perubahan paradigma desain dari struktur kode yang kaku dan saling bergantung menjadi struktur yang lebih modular, fleksibel, dan terorganisasi. Kode sumber tidak hanya menjadi lebih mudah dipahami, tetapi juga lebih siap menghadapi perubahan dan pengembangan di masa depan. Temuan ini memperkuat teori dalam rekayasa perangkat lunak yang menyatakan bahwa kualitas desain memiliki peran sentral dalam meningkatkan maintainability, serta menegaskan bahwa penerapan prinsip desain yang tepat merupakan investasi jangka panjang dalam pengembangan perangkat lunak yang berkelanjutan.

Analisis Dampak Penerapan terhadap Maintainability Kode Sumber

Hasil perbandingan kondisi kode sumber sebelum dan sesudah penerapan prinsip SOLID dan *design patterns* menunjukkan adanya peningkatan maintainability yang signifikan. Kode sumber yang telah melalui proses *refactoring* menjadi lebih mudah dipahami, dianalisis, diuji, dan dimodifikasi. Perubahan ini tidak hanya bersifat visual atau struktural, tetapi juga mencerminkan perbaikan mendasar pada kualitas desain perangkat lunak. Menurut teori kualitas perangkat lunak, maintainability sangat dipengaruhi oleh tingkat kompleksitas, modularitas, dan konsistensi struktur kode. Dengan menurunnya kompleksitas dan meningkatnya modularitas, beban kognitif pengembang dalam memahami sistem menjadi lebih rendah, sehingga proses pemeliharaan dapat dilakukan secara lebih efisien (Maulana, M. I., Sabrina, P. N., & Ramadhan, E. 2024).

Penurunan kompleksitas dan berkurangnya ketergantungan antar modul memberikan dampak positif yang nyata terhadap kemampuan sistem dalam menghadapi perubahan. Berdasarkan konsep *changeability* dan *modifiability* dalam standar kualitas perangkat lunak, sistem yang memiliki ketergantungan rendah akan lebih mudah disesuaikan tanpa menimbulkan

dampak luas pada bagian lain dari sistem. Temuan ini menunjukkan bahwa penerapan prinsip SOLID dan *design patterns* mampu meningkatkan fleksibilitas desain, sehingga perubahan atau penambahan fitur dapat dilakukan dengan risiko kesalahan yang lebih kecil dan waktu pemeliharaan yang lebih singkat (Putraadinatha, I. G. B. V., Suwawi, D. D. J., & Puspitasari, S. Y. 2021).

Meskipun hasil *refactoring* menunjukkan adanya peningkatan jumlah kelas dan baris kode akibat pemisahan tanggung jawab dan penggunaan abstraksi, kondisi ini tidak berdampak negatif terhadap maintainability. Sebaliknya, peningkatan jumlah kelas justru mencerminkan desain yang lebih modular dan terstruktur (Hilalludin., 2025). Dalam teori rekayasa perangkat lunak, jumlah baris kode bukanlah indikator utama kualitas, melainkan bagaimana kode tersebut diorganisasikan dan dirancang. Kode yang lebih panjang namun terstruktur dengan baik akan lebih mudah dipahami dan dipelihara dibandingkan kode yang lebih singkat tetapi kompleks dan saling bergantung.

Dari perspektif *maintainability metrics*, peningkatan kualitas desain ini umumnya ditandai dengan meningkatnya nilai *Maintainability Index* serta menurunnya nilai *Cyclomatic Complexity* dan *Coupling Between Objects*. Meskipun metrik kuantitatif tersebut tidak disajikan secara rinci dalam pembahasan ini, perubahan karakteristik struktur kode yang diamati menunjukkan kecenderungan yang selaras dengan peningkatan nilai maintainability secara keseluruhan. Hal ini memperkuat pandangan bahwa penerapan prinsip desain yang baik memberikan dampak langsung terhadap kualitas pemeliharaan perangkat lunak (Fowler, M. 2023).

Secara keseluruhan, temuan penelitian ini memberikan bukti empiris bahwa maintainability tidak hanya ditentukan oleh aspek ukuran kode, tetapi terutama oleh kualitas desain dan arsitektur perangkat lunak. Penerapan

prinsip SOLID dan *design patterns* terbukti mampu mengatasi permasalahan rendahnya maintainability yang diuraikan pada latar belakang penelitian, dengan cara mengurangi kompleksitas, meningkatkan modularitas, dan memperbaiki struktur kode sumber (Bass, L., Clements, P., & Kazman, R. 2023). Dengan demikian, hasil penelitian ini menegaskan bahwa penerapan prinsip desain yang tepat merupakan strategi yang efektif dan berkelanjutan dalam meningkatkan kualitas perangkat lunak, khususnya dari aspek maintainability.

KESIMPULAN

Berdasarkan hasil dan pembahasan yang telah diuraikan, dapat disimpulkan bahwa penerapan prinsip SOLID dan design patterns memberikan pengaruh yang signifikan terhadap peningkatan maintainability kode sumber perangkat lunak. Refactoring yang dilakukan dengan mengacu pada prinsip Single Responsibility, Dependency Inversion, dan Interface Segregation terbukti mampu menghasilkan struktur kode yang lebih modular, terorganisasi, dan mudah dipahami. Selain itu, pemanfaatan design patterns seperti Factory dan Strategy memperkuat fleksibilitas desain sistem, sehingga perubahan maupun penambahan fitur dapat dilakukan tanpa menimbulkan dampak besar pada bagian kode lainnya. Temuan ini menunjukkan bahwa pendekatan desain berorientasi objek yang baik tidak hanya berdampak pada kualitas teknis kode, tetapi juga pada efisiensi proses pengembangan dan pemeliharaan perangkat lunak.

Lebih lanjut, penelitian ini menegaskan bahwa maintainability tidak semata-mata ditentukan oleh jumlah baris kode atau kompleksitas visual, melainkan oleh kualitas desain dan keteraturan struktur kode. Meskipun penerapan prinsip SOLID dan design patterns dapat meningkatkan jumlah kelas dan abstraksi, hal tersebut justru memperbaiki keterbacaan, kemudahan pengujian, serta keberlanjutan pengembangan sistem dalam jangka panjang.

Dengan demikian, hasil penelitian ini memberikan kontribusi teoretis dan praktis bahwa penerapan prinsip SOLID dan design patterns merupakan solusi yang efektif dalam mengatasi permasalahan rendahnya maintainability kode sumber, sebagaimana diidentifikasi pada latar belakang penelitian, serta layak direkomendasikan sebagai praktik terbaik dalam pengembangan perangkat lunak modern.

DAFTAR PUSTAKA

Alhunaiti, S. A. B. A., & Khan, M. S. (2023). The impact of design patterns on software quality and maintainability. Journal of Student Research. (Jurnal Penelitian Mahasiswa)

Kevin, A., & Kurniawan, H. C. (2025). Analisis pengaruh design pattern terhadap pemeliharaan perangkat lunak Learning Management System. Jurnal Telematika. (journal.ithb.ac.id)

Rhamadhan, W. S., Daniyudin, J., El Sofya, D. R. Z., Hakim, L., & Hakim, L. (2025). Implementasi metode PXP dan prinsip SOLID untuk integrasi modul pada SIMPUSWANGI Banyuwangi. SESSION: Software Dev & Eng. (jurnal.poliwangi.ac.id)

Azhizi, M. H., & Yaqin, M. A. (2024). Analisis penggunaan pemrograman berorientasi objek terhadap maintainability perangkat lunak menggunakan ODOO. Journal Automation Computer Information System, 4(2), 50–59. (jacis.pubmedia.id)

Saad, A. (2025). Enhancing software development efficiency: The role of design patterns in code reusability and flexibility. International Journal of Engineering, Business and Management, 9(1), 81–88. (aipublications.com)

Dharmayanti, D., & Bachtiar, A. M. (2025). Implementation of clean code and design pattern to improve maintainability in content marketing application. AIP Conference Proceedings, 3200, 040018. (simlitabmas.unikom.ac.id)

Rochimah, S., Akbar, R. J., & Langsari, K. (2025). Investigating design patterns' impact on application performance and complexity. IPTEK The Journal for Technology and Science. (iptek.its.ac.id)

Ramachandrappa, N. C. (2024). SOLID design principles in software engineering. International Journal of Computer Trends and Technology, 72(9), 18–23. (Seventh Sense Research Group®)

Maulana, M. I., Sabrina, P. N., & Ramadhan, E. (2024). Peningkatan kualitas efficiency dan maintainability pada sistem informasi web sekolah dengan refactoring. Seminar Nasional CORISINDO 2024. (corisindo.utb-univ.ac.id)

Putraadinatha, I. G. B. V., Suwawi, D. D. J., & Puspitasari, S. Y. (2021). Pengaruh design pattern terhadap maintainability aplikasi mobile. eProceedings of Engineering. (openlibrarypublications.telkomuniversity.ac.id)

Shrestha, J. (2025). Evaluating the application of SOLID principles in modern AI framework architectures. arXiv preprint arXiv:2503.13786. (arXiv)

Wang, Z., Ling, R., Wang, C., Yu, Y., Li, Z., Xiong, F., & Zhang, W. (2025). MaintainCoder: Maintainable code generation under dynamic requirements. arXiv preprint arXiv:2503.24260. (arXiv)

Applying SOLID principles for the refactoring of legacy code (2024). Journal of Systems and Software. (ScienceDirect)

Understanding the importance of design patterns in software development (2025). SSRN Electronic Journal. (SSRN)

Martin, R. C. (2024). Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall.

Fowler, M. (2023). Refactoring: Improving the Design of Existing Code (2nd ed.). Addison-Wesley.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (2021). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.

Schmidt, D. C., Stal, M., Rohnert, H., & Buschmann, F. (2022). Pattern-Oriented Software Architecture Volume 2: Patterns for Concurrent and Networked Objects. Wiley.

Bass, L., Clements, P., & Kazman, R. (2023). Software Architecture in Practice (4th ed.). Addison-Wesley.

Zohri, M. H., & Hilalludin, H. (2025). Pemikiran Ibn Jinni Tentang Linguistik Arab Dan Relevansinya Bagi Kajian Linguistik. Qawa'id: Jurnal Bahasa Dan Sastra Arab, 1(01), 25-35.

Sugari, D., & Hilalludin, H. (2025). Kontribusi Psikologi Perkembangan dalam Strategi Pembelajaran di Sekolah. Jurnal Ar-Ruhul Ilmi: Jurnal Pendidikan Dan Pemikiran Islam, 1(01), 47-61.

Saputra, J., Hilalludin, H., & Gibran, I. R. (2024). Peran Kepemimpinan Kepala Sekolah dan Profesionalisme Guru Dalam Meningkatkan Mutu Pendidikan Indonesia. Jurnal Pendidikan Dan Ilmu Sosial (Jupendis), 2(4), 163-172.

Sugari, D., & Hilalludin, H. (2025). Transformasi Pendidikan di Era Digital Peluang danTantangan bagi Generasi Muda. LUXFIA: Journal of Multidisciplinary Research, 1(1), 57-68.

Sugari, D., & Hilalludin, H. (2025). Optimalisasi Fungsi Masjid Sebagai Pusat Ibadah, Pendidikan, dan Sosial Masyarakat Melalui Program Pengabdian di Masjid Al-

Muttaqin Semin, Gunungkidul. IQOMAH: Jurnal Pengabdian Kepada Masyarakat, 1(01), 50-63.

Sugari, D., & Hilalludin, H. (2025). Peran Maqashid Syariah dalam Pengembangan Produk Perbankan Islam yang Berkelanjutan. AL HILALI: Jurnal Perbankan Dan Ekonomi Islam, 1(1), 01-15.