Merkurius: Jurnal Riset Sistem Informasi dan Teknik Informatika
Volume 3, Nomor 4, Juli 2025
e-ISSN: 3031-8912; p-ISSN: 3031-8904; Hal. 193-199
DOI: https://doi.org/10.61132/merkurius.v3i4.928
Available online at: https://journal.arteii.or.id/index.php/Merkurius

Analisis Efektivitas Konsep Polimorfisme dalam Menyederhanakan Desain Perangkat Lunak

Naurah Atikah Nurpadhilah[1*], Aliya Dwi Ardiyanti[2], M. Aufa Rafiqi[3], Nur Ayu Siti Hardianti[4], Mufid Faruq Aziz[5]
[1-5] Universitas Muhammadiyah Bengkulu, Bengkulu, Indonesia

Alamat: Jalan Bali Kota Bengkulu 38119, Kec. Teluk Segara, Kota Bengkulu
Korespondensi penulis: naurahatikah.17@gmail.com

Abstract. *Object-Oriented Programming (OOP) is a programming paradigm that emphasizes the use of objects and classes to build modular and maintainable software. One of the fundamental concepts in OOP is polymorphism, which is the ability of objects from different classes to respond to the same method in different ways. This study aims to analyze the effectiveness of the polymorphism concept in directing software design in terms of code structure, development utility, and maintenance efficiency. The research method used is a literature study and program code simulation using a procedural approach compared to an object-oriented approach that applies runtime polymorphism. The results of the analysis show that the use of polymorphism can significantly reduce the number of lines of code, logical structure traffic, and facilitate feature expansion without the need to change existing code. Thus, polymorphism has proven effective in improving readability, reusability, and scalability of software design.*

Keywords: *Object-Oriented Programming, Polymorphism, Software Design, OOP, Efficient Code*

Abstrak. Pemrograman Berorientasi Objek (PBO) merupakan paradigma pemrograman yang menekankan penggunaan objek dan kelas untuk membangun perangkat lunak yang modular dan mudah dipelihara. Salah satu konsep fundamental dalam PBO adalah polimorfisme, yaitu kemampuan objek dari berbagai kelas untuk merespons metode yang sama dengan cara yang berbeda. Penelitian ini bertujuan untuk menganalisis efektivitas konsep polimorfisme dalam menyederhanakan desain perangkat lunak dari sisi struktur kode, fleksibilitas pengembangan, dan efisiensi pemeliharaan. Metode penelitian yang digunakan adalah studi literatur dan simulasi kode program menggunakan pendekatan prosedural dibandingkan dengan pendekatan berorientasi objek yang menerapkan *runtime polymorphism*. Hasil analisis menunjukkan bahwa penggunaan polimorfisme dapat secara signifikan mengurangi jumlah baris kode, menyederhanakan struktur logika, serta mempermudah ekspansi fitur tanpa perlu mengubah kode yang sudah ada. Dengan demikian, polimorfisme terbukti efektif dalam meningkatkan keterbacaan, reusabilitas, dan skalabilitas desain perangkat lunak.

Kata kunci: Pemrograman Berorientasi Objek, Polimorfisme, Desain Perangkat Lunak, OOP, Efisiensi Kode

1. PENDAHULUAN

Latar Belakang

Dalam era perkembangan teknologi perangkat lunak yang sangat pesat, pendekatan terhadap desain perangkat lunak yang efisien, fleksibel, dan mudah dipelihara menjadi sangat penting. Salah satu prinsip utama dalam pemrograman berorientasi objek (OOP) yang mendukung tujuan tersebut adalah polimorfisme. Polimorfisme memungkinkan objek dari kelas yang berbeda untuk diakses melalui antarmuka yang sama, sehingga mampu meningkatkan keterbacaan kode, mengurangi redundansi, dan memudahkan pengembangan maupun pemeliharaan sistem.

Menurut Sommerville (2020), polimorfisme berkontribusi besar dalam fleksibilitas desain perangkat lunak, karena memungkinkan penggunaan tipe-tipe yang berbeda secara

Received: Mei 19, 2025; Revised: Juni 30, 2025; Accepted: Juli 05, 2025; Published: Juli 07, 2025

umum tanpa mengubah struktur logika program. Hal ini memungkinkan programmer untuk membuat sistem yang lebih modular dan adaptif terhadap perubahan.

Sementara itu, Sharma & Kaur (2019) menyatakan bahwa penggunaan polimorfisme dapat mempercepat proses pengembangan perangkat lunak karena struktur kode yang lebih ringkas dan *reuseable*. Dalam konteks tim pengembang perangkat lunak, hal ini juga dapat mempermudah kolaborasi karena pengembang tidak perlu memahami seluruh implementasi suatu kelas untuk menggunakannya secara efektif.

Dalam praktiknya, konsep ini terbagi ke dalam dua bentuk utama: polimorfisme statis (*compile-time*) dan polimorfisme dinamis (*run-time*). Menurut Patil dan Kulkarni (2021), polimorfisme dinamis lebih unggul dalam desain sistem berskala besar karena mendukung prinsip substitusi dan late binding, yang menjadikan sistem lebih terbuka untuk ekspansi namun tertutup terhadap perubahan (prinsip *Open/Closed* dari SOLID).

Namun demikian, efektivitas polimorfisme dalam penyederhanaan desain perangkat lunak masih sering diperdebatkan, terutama dalam konteks efisiensi kinerja dan kompleksitas *debugging*. Oleh karena itu, penelitian ini bertujuan untuk menganalisis secara lebih mendalam bagaimana konsep polimorfisme dapat digunakan secara efektif dalam menyederhanakan desain perangkat lunak, berdasarkan studi literatur, studi kasus, dan pendekatan teoritis.

2. METODE PENELITIAN

Penelitian ini menggunakan metode deskriptif kualitatif, yaitu metode yang bertujuan untuk mendeskripsikan dan menganalisis fenomena tertentu secara mendalam berdasarkan data non-numerik. Fokus dari penelitian ini adalah menganalisis efektivitas konsep polimorfisme dalam menyederhanakan desain perangkat lunak melalui pendekatan kajian pustaka, analisis kode program, dan studi kasus sederhana.

Objek dalam penelitian ini adalah konsep polimorfisme dalam pemrograman berorientasi objek, khususnya dalam konteks bahasa pemrograman Java. Polimorfisme yang dikaji mencakup:

1. *Compile-time polymorphism (method overloading)*
2. *Runtime polymorphism (method overriding)*

Penelitian ini juga mencakup struktur desain perangkat lunak yang menerapkan dan tidak menerapkan konsep polimorfisme untuk dibandingkan efektivitasnya.

Teknik pengumpulan data dilakukan melalui dua pendekatan utama:

1. Studi Literatur
   a) Mengkaji teori-teori dan definisi dari buku, jurnal, dan artikel ilmiah yang relevan tentang PBO dan polimorfisme.
   b) Mengambil referensi dari pakar OOP seperti *Grady Booch, Deitel & Deitel, Pressman,* dan sumber lain yang kredibel.
2. Simulasi dan Studi Kasus Kode
   a) Melakukan implementasi dua versi kode program: satu menggunakan pendekatan prosedural dan satu menggunakan pendekatan OOP dengan polimorfisme.
   b) Menganalisis struktur, jumlah baris kode, tingkat fleksibilitas, dan kemudahan ekspansi sistem dari masing-masing versi.

Data yang dikumpulkan akan dianalisis dengan metode analisis perbandingan. Langkah-langkahnya meliputi:

1. Menyusun dua buah potongan kode dengan struktur dan tujuan yang sama, tetapi dengan pendekatan berbeda (dengan dan tanpa polimorfisme).
2. Mengukur indikator efektivitas, seperti:
   a) Jumlah baris kode
   b) Kemudahan ekspansi fitur
   c) Tingkat keterbacaan dan kompleksitas logika
   d) Kemudahan pemeliharaan *(maintenance)*

Analisis dilakukan secara deskriptif, yaitu dengan menjelaskan hasil dari kedua pendekatan, menguraikan kelebihan dan kekurangannya, lalu menyimpulkan efektivitas penggunaan konsep polimorfisme dalam desain perangkat lunak.

3. HASIL DAN PEMBAHASAN

Hasil Simulasi Kode Program :

Untuk menganalisis efektivitas konsep polimorfisme, dilakukan eksperimen kode dengan membuat dua versi program sederhana dalam bahasa *Java:*

1. Versi A (Tanpa Polimorfisme): menggunakan pendekatan prosedural dengan logika ***if-else.***
2. Versi B (Dengan Polimorfisme): menggunakan pendekatan *object-oriented programming* dengan *method overriding.*

Studi Kasus: Sistem Suara Hewan

1. Versi A – Tanpa Polimorfisme *(Procedural approach)*

Gambar 1. *Procedural approach*

2. Versi B – Dengan Polimorfisme (OOP *approach*)

Gambar 2. OOP *approach*

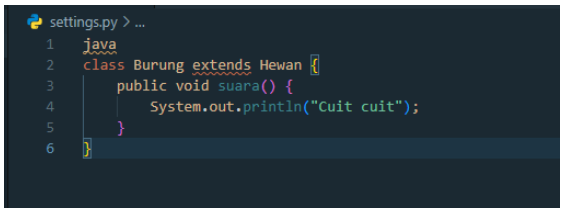
Efektivitas dalam Struktur Kode

Penerapan polimorfisme menghasilkan struktur kode yang lebih ringkas, modular, dan lebih mudah dipahami. Kode tidak perlu menggunakan ***if-else*** untuk membedakan perilaku setiap objek, cukup dengan membuat subclass dan meng-override metode sesuai kebutuhan.

Kemudahan Ekspansi Sistem

Salah satu keunggulan utama dari polimorfisme adalah pada ekstensibilitas sistem. Jika dalam waktu mendatang diperlukan jenis hewan baru, pada versi prosedural pengembang harus memodifikasi logika ***if-else***. Sedangkan pada versi OOP, pengembang hanya perlu membuat kelas baru yang mewarisi kelas **Hewan**.

Contoh penambahan jenis **Burung:**



Gambar 3. Penambahan jenis Burung dengan versi OOP

Tidak ada perubahan pada struktur utama program, sehingga mematuhi prinsip *open-closed*.

Konsistensi dan Pemeliharaan

Kode yang menggunakan polimorfisme lebih konsisten dan mudah dipelihara. Setiap perilaku diatur di dalam kelas masing-masing. Jika terdapat perubahan suara untuk **Kucing**, maka cukup ubah kelas **Kucing** tanpa mempengaruhi bagian kode lain.

Keterbatasan Polimorfisme

Meskipun polimorfisme memberikan banyak manfaat, penggunaannya tetap memerlukan pemahaman tentang struktur pewarisan dan antarmuka. Jika tidak dirancang dengan baik, sistem bisa menjadi terlalu abstrak dan membingungkan, terutama bagi pemula.

Interpretasi Temuan

Berdasarkan eksperimen kode, ditemukan bahwa:

1. Polimorfisme menyederhanakan desain perangkat lunak dari sisi struktur dan logika program.
2. Waktu pengembangan dan biaya pemeliharaan dapat ditekan dengan pendekatan ini, karena perubahan tidak menimbulkan efek berantai.
3. Prinsip desain perangkat lunak modern seperti SOLID dan *Clean Code* dapat diterapkan lebih mudah melalui polimorfisme.

4. KESIMPULAN DAN SARAN

Berdasarkan hasil analisis dan simulasi yang dilakukan, dapat disimpulkan bahwa konsep polimorfisme dalam pemrograman berorientasi objek terbukti efektif dalam menyederhanakan desain perangkat lunak, khususnya dalam hal struktur kode, fleksibilitas pengembangan, dan efisiensi pemeliharaan. Penerapan polimorfisme memungkinkan pengembangan sistem yang lebih modular, terbuka terhadap ekspansi (*open for extension*), dan tertutup terhadap perubahan (*closed for modification*), sesuai dengan prinsip desain perangkat lunak yang baik. Melalui pendekatan ini, jumlah baris kode dapat diminimalkan, kompleksitas logika program

berkurang, dan penambahan fitur baru dapat dilakukan tanpa mengganggu struktur kode yang telah ada.

Namun demikian, efektivitas polimorfisme sangat bergantung pada pemahaman yang baik terhadap struktur pewarisan, *interface*, serta prinsip-prinsip desain objek. Jika diterapkan secara tidak tepat, polimorfisme justru dapat menyebabkan kode yang terlalu abstrak, sulit dilacak, dan menantang untuk diuji, terutama bagi pengembang yang masih kurang berpengalaman.

Berdasarkan hasil analisis yang telah dilakukan, disarankan kepada para pengembang perangkat lunak untuk memahami secara mendalam konsep polimorfisme sebelum mengimplementasikannya dalam proyek nyata. Penggunaan polimorfisme hendaknya diintegrasikan secara seimbang dengan prinsip desain lain seperti SOLID dan *Clean Code* agar manfaatnya dalam menyederhanakan desain dapat tercapai secara maksimal. Penggunaan dokumentasi dan pemisahan tanggung jawab kelas juga penting agar sistem tetap terbaca dan mudah dipelihara. Selain itu, bagi pengembang pemula, pelatihan dan praktik penerapan konsep ini dalam skala kecil perlu dilakukan terlebih dahulu untuk menghindari penerapan yang berlebihan atau tidak tepat sasaran. Penelitian lanjutan juga disarankan untuk menguji efektivitas polimorfisme dalam skenario proyek berskala besar dan berorientasi tim guna mengevaluasi kontribusinya terhadap kolaborasi dan pengelolaan kompleksitas kode dalam pengembangan perangkat lunak *modern*.

DAFTAR REFERENSI

Albahari, J., & Albahari, B. (2022). C# 10 in a nutshell: The definitive reference (1st ed.). O'Reilly Media.

Booch, G. (2007). Object-oriented analysis and design with applications (3rd ed.). Addison-Wesley.

Deitel, P. J., & Deitel, H. M. (2017). Java: How to program (10th ed.). Pearson Education.

Fowler, M. (2004). UML distilled: A brief guide to the standard object modeling language (3rd ed.). Addison-Wesley.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design patterns: Elements of reusable object-oriented software. Addison-Wesley.

Larman, C. (2012). Applying UML and patterns: An introduction to object-oriented analysis and design and iterative development (3rd ed.). Prentice Hall.

Martin, R. C. (2003). Agile software development: Principles, patterns, and practices. Prentice Hall.

Oracle. (2023). The Java™ Tutorials. Oracle. https://docs.oracle.com/javase/tutorial/

Patil, A., & Kulkarni, D. (2021). A comparative study of static and dynamic polymorphism in object-oriented programming. International Journal of Computer Applications, 183(25), 1–6. https://doi.org/10.5120/ijca2021921182

Pressman, R. S. (2015). Software engineering: A practitioner's approach (8th ed.). McGraw-Hill Education.

Sharma, R., & Kaur, A. (2019). Role of polymorphism in software development using OOP. International Journal of Advanced Research in Computer Science, 10(4), 54–58. https://doi.org/10.26483/ijarcs.v10i4.6460

Sommerville, I. (2020). Software engineering (10th ed.). Pearson Education.