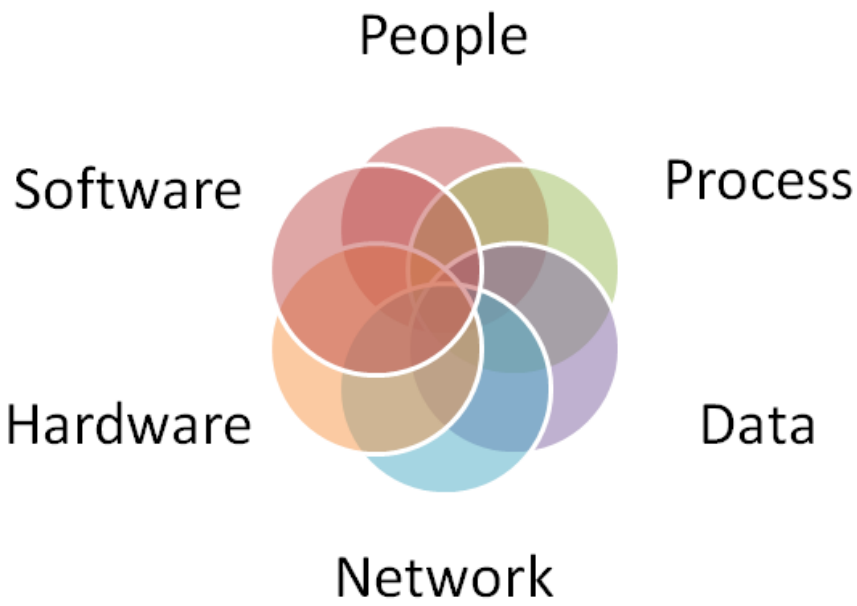
OAJIS
Open Access
Journal of
Information
Systems
is.its.ac.id/pubs/oajis/

ISSN 1979-3979

jurnal sisfo

Inspirasi Profesional Sistem Informasi

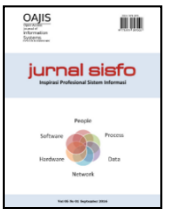


Vol. 11 No. 02 Oktober 2024

OAJIS
Open Access
Journal of
Information
Systems
is.its.ac.id/pubs/oajis/

jurnal sisfo

Jurnal Sisfo Vol. 11 No. 02 (2024)



Sekretariat

Departemen Sistem Informasi – Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember (ITS) – Surabaya
Telp. 031-5999944 Fax. 031-5964965
Email: editor@jurnalsisfo.org
Website: http://jurnalsisfo.org

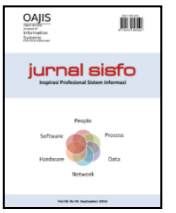
Jurnal SISFO juga dipublikasikan di *Open Access Journal of Information Systems* (OAJIS)

Website: http://is.its.ac.id/pubs/oajis/index.php

OAJIS
Open Access
Journal of
Information
Systems
is.its.ac.id/pubs/oajis/

jurnal sisfo

Jurnal Sisfo Vol. 11 No. 02 (2024)



Mitra Bestari

Prof. Erma Suryani, S.T., M.T., Ph.D. (Institut Teknologi Sepuluh Nopember)

Prof. Dr. Wiwik Anggraeni, S.Si., M.Kom. (Institut Teknologi Sepuluh Nopember)

Tony Dwi Susanto, S.T., M.T., Ph.D. (Institut Teknologi Sepuluh Nopember)

Yogantara Setya Dharmawan, S.Kom, M.BusProcessMgt. (Institut Teknologi Sepuluh Nopember)

Arif Wibisono, S.Kom., M.Sc., Ph.D. (Institut Teknologi Sepuluh Nopember)

Dr. Bambang Setiawan, S.Kom., M.T. (Institut Teknologi Sepuluh Nopember)

Dr. Muhammad Ainul Yaqin, S.T., M.Kom. (Universitas Islam Negeri Maulana Malik Ibrahim)

Taufik, S.T., M.Kom. (Universitas Airlangga)

Dr. Apol Pribadi Subriadi, S.T., M.T. (Institut Teknologi Sepuluh Nopember)

Muhammad Amirul Haq, S.T., M.Sc. (Universitas Muhammadiyah Surabaya)

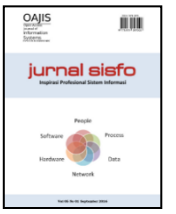
Dhiani Tresna Absari, S.T., M.Kom. (Universitas Surabaya)

Dr. Mudjahidin, S.T., M.T. (Institut Teknologi Sepuluh Nopember)

OAJIS
Open Access
Journal of
Information
Systems
is.its.ac.id/pubs/oajis/

jurnal sisfo

Jurnal Sisfo Vol. 11 No. 02 (2024)



Daftar Isi

OAJIS
Open Access
Journal of
Information
Systems
is.its.ac.id/pubs/oajis/

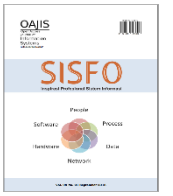
jurnal sisfo

Jurnal Sisfo Vol. 11 No. 2 (2024)

Halaman ini sengaja dikosongkan

OAJIS
Open Access Journal of Information Systems
is.its.ac.id/pubs/oajis/

SISFO
Inspirasi Profesional Sistem Informasi
Jurnal SISFO Vol.11 No.02 (2024) 85–94



Analisis Penerapan Prinsip SOLID pada Tugas Proyek Mahasiswa UIN Malang

Alfred Rajendra Wijaya , Ela Ilmatul Hidayah* , M. Ainul Yaqin

Program Studi Teknik Informatika Fakultas Sains dan Teknologi, UIN Maulana Malik Ibrahim, Jalan Gajayana 50, Malang

Abstract

The use of artificial intelligence (AI) based on Large Language Models (LLMs), such as GPT-4, GPT-3.5, and Gemini, is becoming more prevalent in software development, particularly in identifying and refactoring code to adhere to SOLID design principles. This study analyzes the effectiveness and accuracy of AI in detecting and refactoring code to follow these principles. We used code samples from second-semester Informatics Engineering students at UIN Malang as the dataset. The analysis was performed by comparing the SOLID principle detection results from GPT-4, GPT-3.5, and Gemini. The findings show that Gemini is the most accurate in identifying and applying SOLID principles, achieving a perfect score compared to GPT-4 and GPT-3.5, which had lower accuracy levels.

Keywords: SOLID Principles, Artificial Intelligence, GPT-4, GPT-3.5, Gemini

Abstrak

Penggunaan kecerdasan buatan (AI) berbasis *Large Language Models* (LLM), seperti GPT-4, GPT-3.5, dan Gemini, semakin umum dalam pengembangan perangkat lunak, terutama dalam mengidentifikasi dan merefaktor kode agar sesuai dengan prinsip desain SOLID. Studi ini menganalisis efektivitas dan akurasi AI dalam mendeteksi dan merefaktor kode agar mengikuti prinsip-prinsip tersebut. Kami menggunakan sampel kode dari mahasiswa Teknik Informatika semester dua di UIN Malang sebagai dataset. Analisis dilakukan dengan membandingkan hasil deteksi prinsip SOLID dari GPT-4, GPT-3.5, dan Gemini. Temuan menunjukkan bahwa Gemini adalah yang paling akurat dalam mengidentifikasi dan menerapkan prinsip SOLID, mencapai skor sempurna dibandingkan dengan GPT-4 dan GPT-3.5 yang memiliki tingkat akurasi lebih rendah.

Kata kunci: SOLID *Principles, Artificial Intelligence,* GPT-4, GPT-3.5, Gemini

Histori Artikel: Disubmit 25-06-2024; Direvisi 18-09-2024; Diterima 01-10-2024; Tersedia online 30-06-2024

*Corresponding Author
Email address: elailma217@gmail.com (Ela Ilmatul Hidayah)
https://doi.org/10.24089/j.sisfo.2024.06.007

1. Pendahuluan

Pengembangan perangkat lunak telah menjadi bagian integral dari kehidupan modern dan memainkan peran krusial dalam berbagai aspek kehidupan kita [1]. Dari aplikasi bisnis hingga perangkat rumah pintar, perangkat lunak mempengaruhi hampir setiap industri. Namun, semakin kompleksnya tuntutan pasar menantang para pengembang untuk memastikan kualitas dan efisiensi kode yang dihasilkan. Dalam era digital yang terus berubah, pengembangan perangkat lunak menjadi pusat inovasi dan kemajuan. Namun, kompleksitas sistem yang semakin meningkat—dengan ratusan ribu hingga jutaan baris kode dan berbagai fitur—menimbulkan tantangan dalam pemahaman, pemeliharaan, dan manajemen kode. Di sinilah prinsip SOLID (*Single Responsibility*, *Open/Closed*, *Liskov Substitution*, *Interface Segregation*, dan *Dependency Inversion*) hadir sebagai panduan desain yang membantu pengembang merancang sistem yang mudah dimengerti, mudah dipelihara, dan mudah dikembangkan. Dalam penelitian ini, kami mengeksplorasi penerapan prinsip SOLID dan menguji efektivitasnya melalui prototipe fungsional.

Prinsip SOLID (*Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation*, dan *Dependency Inversion*) adalah seperangkat pedoman desain yang dikembangkan untuk membantu pengembang perangkat lunak merancang sistem yang mudah dimengerti, mudah dipelihara, dan mudah dikembangkan. Prinsip SOLID menghasilkan desain yang fleksibel, minim ketergantungan, dan kemudahan perawatan yang lebih tinggi [2]. Prinsip-prinsip ini pertama kali diperkenalkan oleh Robert C. Martin, seorang ahli dalam pengembangan perangkat lunak, dan telah menjadi dasar dalam pengembangan perangkat lunak berorientasi objek. Tinjauan prinsip-prinsip "SOLID" diberikan bersama dengan pemeriksaan empiris prinsip-prinsip ini berdasarkan penggunaan prototipe fungsional.

Pesatnya perkembangan teknologi informasi berbasis komputer telah membuat banyak perubahan dalam kehidupan manusia, salah satunya adalah teknologi *Artificial Intelligence* [3]. Kecerdasan buatan telah digunakan secara luas untuk mengidentifikasi pola dalam kode yang rumit dan membantu dalam *refactoring* kode untuk mematuhi prinsip-prinsip desain yang baik seperti prinsip SOLID. Sebagai contoh, dalam penelitian yang dilakukan oleh Johnson et al. (2020), AI berhasil digunakan untuk secara otomatis mendeteksi pelanggaran prinsip *Open/Closed* dalam aplikasi yang besar dengan tingkat akurasi yang tinggi [4]. Pada penelitian ini menunjukkan bahwa AI tidak hanya bisa melakukan identifikasi masalah desain tetapi juga memberikan solusi refactoring yang bisa diterapkan.

AI adalah teknologi yang mengambil alih kapasitas intelektual yang sebelumnya hanya dikaitkan dengan manusia. AI melibatkan penggunaan algoritma dan model matematika untuk memungkinkan komputer dan sistem lainnya untuk belajar dari data, mengenali pola, dan membuat keputusan yang cerdas [5]. Pernyataan ini menyiratkan bahwa penerapan AI dalam rekayasa perangkat lunak tidak hanya mempercepat proses pengembangan dan pengujian, tetapi juga meningkatkan kualitas perangkat lunak yang dihasilkan.

Studi oleh Gupta dan Sharma menemukan bahwa penggunaan algoritma AI dalam *refactoring* kode meningkatkan efisiensi pengembangan sebesar 30% dibandingkan dengan metode manual. Algoritma ini, ketika diterapkan pada database kode yang luas, juga menunjukkan penurunan 25% dalam jumlah *bug* yang dilaporkan dalam enam bulan setelah implementasi [6]. Statistik ini membuktikan bahwa AI tidak hanya mempercepat proses *refactoring* tetapi juga meningkatkan kualitas keseluruhan kode.

Teknologi kecerdasan buatan memiliki banyak keuntungan yang dapat meningkatkan kehidupan manusia. Salah satu keuntungan terbesar adalah kemampuannya untuk mengambil keputusan dengan cepat dan akurat berdasarkan data serta dapat membantu meningkatkan efisiensi di berbagai sektor. Hal ini dapat membantu mengurangi kesalahan manusia dan mempercepat waktu respon dalam situasi kritis [7].

Meskipun kecerdasan buatan menawarkan banyak keuntungan, penerapannya dalam pengembangan perangkat lunak juga menghadapi beberapa tantangan serius. Misalnya, Zhang et al. menyoroti bahwa salah

satu tantangan terbesar adalah kebutuhan akan dataset pelatihan yang besar dan berkualitas tinggi untuk memastikan AI dapat belajar secara efektif tanpa bias [8]. Selain itu, masalah keamanan seperti risiko kebocoran data dan potensi untuk generasi kode berbahaya tidak boleh diabaikan. Penerapan AI juga membutuhkan pengawasan yang ketat untuk memastikan bahwa saran yang dihasilkan tidak merusak arsitektur sistem yang sudah ada atau mengintroduksi kesalahan baru. Pengguna dan peneliti harus menyadari bahwa kinerja ChatGPT mungkin terhambat saat menganalisis sumber yang kompleks [9].

Dengan fokus pada penerapan prinsip SOLID, penelitian ini bertujuan untuk mengeksplorasi kemampuan GPT-4 dalam Analisa kode program yang menggunakan arsitektur SOLID. Penelitian ini didorong oleh pertanyaan penelitian utama "Bagaimana kecerdasan buatan dapat digunakan untuk menganalisa dan memperbaiki prinsip SOLID dalam pengembangan perangkat lunak?". Analisis ini penting untuk memahami potensi AI dalam meningkatkan kualitas dan efisiensi proses pengembangan perangkat lunak, serta tantangan yang mungkin dihadapi dalam implementasinya.

Melalui penelitian ini, kami berharap dapat memberikan wawasan baru dan rekomendasi praktis bagi pengembang perangkat lunak yang tertarik menggunakan AI dalam proses desain dan pengembangan. Dengan meningkatnya kompleksitas perangkat lunak, kebutuhan akan solusi otomatis yang dapat mengadopsi dan memelihara prinsip-prinsip desain yang baik menjadi semakin kritikal. Kecerdasan buatan, dengan kapabilitasnya yang luas untuk menganalisis, memodelkan, dan memprediksi pola-pola dalam data menawarkan kemungkinan untuk meningkatkan cara kita mendekati desain perangkat lunak [10]. Algoritma AI yang canggih, tidak hanya dapat mempercepat proses refactoring tetapi juga meningkatkan kualitas kode secara signifikan, mengurangi kemungkinan kesalahan manusia, dan memastikan konsistensi arsitektur yang lebih baik.

2. Tinjauan Pustaka

Implementasi prinsip-prinsip SOLID dalam pengembangan perangkat lunak menjadi krusial untuk memastikan desain yang *maintainable* dan *scalable*. Prinsip-prinsip ini terdiri dari:

2.1 Single Responsibility Principle (SRP)

SRP menekankan bahwa setiap kelas atau modul dalam sebuah program seharusnya hanya memiliki satu alasan untuk berubah. Penelitian oleh Lionel C. Briand, Christian Bunse, dan John W. Daly mengevaluasi bagaimana penerapan SRP dapat meningkatkan kemampuan pemeliharaan sistem perangkat lunak.

2.2 Open/Closed Principle (OCP)

OCP mendorong agar kelas-kelas dalam sistem perangkat lunak dapat di-extend tanpa perlu memodifikasi kode yang sudah ada. Studi yang dilakukan oleh Amit Kumar Shrivastava, Sanjiv Kumar Shriwastava, dan Rohit Sharma dalam "*Evolution* of *Object Oriented Analysis* & *Design in Software Engineering*" menggambarkan evolusi penerapan OCP dalam pendekatan berorientasi objek.

2.3 Liskov Subtitution Principle (LSP)

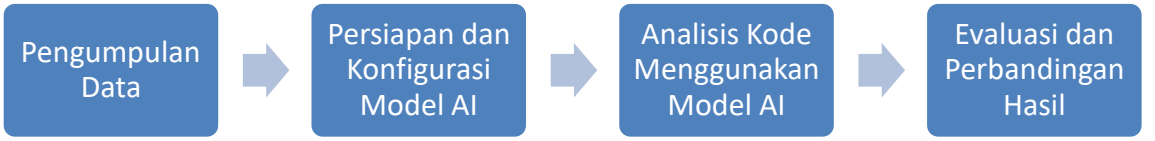
LSP menyatakan bahwa objek dari kelas apapun harus bisa diganti dengan instance dari subkelasnya tanpa mengganggu kebenaran dari program tersebut. Penelitian oleh Tomas Smolik tentang "*An Object-Oriented File System - an Example of Using the Class Hierarchy Framework Concept*" memberikan contoh konkret implementasi LSP dalam pengembangan sistem berkas berorientasi objek [11].

2.4 Interface Segregation Principle (ISP)

ISP menyarankan agar klien hanya bergantung pada antarmuka yang mereka perlukan, bukan seluruh antarmuka sistem. Studi yang mendalam tentang penerapan ISP dalam konteks pengembangan perangkat lunak dapat ditemukan dalam literatur yang membahas desain sistem dengan prinsip-prinsip SOLID.

3. Metodologi

Penelitian ini melibatkan beberapa tahapan yang dirancang untuk menganalisis efektivitas dan akurasi penggunaan kecerdasan buatan dalam mengidentifikasi dan melakukan *refactoring* kode agar sesuai dengan prinsip desain SOLID. Tahapan-tahapan tersebut dapat dilihat pada Gambar 1.



Gambar 1. Metodologi Penelitian

3.1 Pengumpulan Data

Tahap pengumpulan data meliputi pengumpulan data dari kode program yang dihasilkan oleh mahasiswa semester 2 Teknik Informatika UIN Malang untuk diambil sebagai sampel penelitian. Kode-kode ini merupakan hasil tugas yang diberikan selama satu semester yang mana pada program ini relevan dengan penerapan prinsip SOLID dalam pengembangan perangkat lunak. Proses pengumpulan data dilakukan melalui dua cara:

1) Dokumentasi Tugas Mahasiswa: Kode yang digunakan dalam penelitian diambil langsung dari hasil tugas proyek mahasiswa yang dikumpulkan di bawah bimbingan dosen.
2) Studi Pustaka: Peneliti juga mengumpulkan literatur tambahan mengenai prinsip SOLID untuk memperkaya analisis teoritis terhadap kode yang dianalisis.

3.2 Persiapan dan Konfigurasi Model AI

Setelah data terkumpul, tahap berikutnya adalah persiapan dan konfigurasi model kecerdasan buatan (AI). Dalam penelitian ini, digunakan dua model AI yaitu GPT dan Gemini. Adapun langkah-langkah persiapan meliputi:

1) Pemilihan Model AI: GPT dan Gemini dipilih karena memiliki kemampuan untuk memahami dan menganalisis kode pemrograman dengan baik.
2) Konfigurasi Lingkungan Pemodelan: Pengaturan lingkungan komputasi dan penyusunan model dilakukan agar model AI dapat berjalan dengan optimal. Hal ini termasuk instalasi pustaka yang dibutuhkan, serta konfigurasi model agar sesuai dengan spesifikasi tugas yang akan dianalisis.
3) Uji Coba Awal: Setelah konfigurasi, dilakukan uji coba awal pada model untuk memastikan bahwa model siap digunakan untuk menganalisis kode sumber dari tugas mahasiswa.

3.3 Analisis Kode Menggunakan Model AI

Pada tahap ini, kode yang dikumpulkan dianalisis menggunakan model AI yang telah dipersiapkan. Adapun proses analisis kode terdiri dari beberapa langkah:

1) Pemrosesan Kode Sumber: Model GPT dan Gemini memproses kode yang telah dikumpulkan, menganalisis penerapan prinsip SOLID pada kode tersebut.
2) Identifikasi Penerapan Prinsip SOLID: Model AI melakukan identifikasi terhadap penerapan lima prinsip SOLID, yaitu *Single Responsibility Principle, Open/Closed Principle, Liskov Substitution Principle, Interface Segregation Principle*, dan *Dependency Inversion Principle*, dalam kode proyek mahasiswa.
3) Pendeteksian Kesalahan atau Penyimpangan: Model AI juga mendeteksi apakah terdapat penyimpangan atau kesalahan penerapan prinsip SOLID yang berpotensi menurunkan kualitas kode.

3.4 Evaluasi dan Perbandingan Hasil

Pada tahap ini, kode yang dikumpulkan dianalisis menggunakan model AI yang telah dipersiapkan. Adapun proses analisis kode terdiri dari beberapa langkah:

1) Pemrosesan Kode Sumber: Model GPT dan Gemini memproses kode yang telah dikumpulkan, menganalisis penerapan prinsip SOLID pada kode tersebut.
2) Identifikasi Penerapan Prinsip SOLID: Model AI melakukan identifikasi terhadap penerapan lima prinsip SOLID, yaitu *Single Responsibility Principle, Open/Closed Principle, Liskov Substitution Principle, Interface Segregation Principle*, dan *Dependency Inversion Principle*, dalam kode proyek mahasiswa.
3) Pendeteksian Kesalahan atau Penyimpangan: Model AI juga mendeteksi apakah terdapat penyimpangan atau kesalahan penerapan prinsip SOLID yang berpotensi menurunkan kualitas kode.

4. Hasil dan Pembahasan

4.1 Kriteria Penilaian untuk Analisis Prinsip SOLID

Dibawah merupakan bagian daari kriteria penilaian untuk melakukan analisis prinsip SOLID:

4.1.1 *Single Responsibility Principle* (SRP)

Menurut penelitian Chuan Qin (2023), SRP menyatakan bahwa sebuah kelas seharusnya hanya memiliki satu alasan untuk berubah, yang berarti ia harus memiliki satu tanggung jawab. Prinsip ini mendorong desain kelas kecil yang fokus pada satu aspek fungsi, sehingga meningkatkan keterpeliharaan dan fleksibilitas [8]. Semua model AI ditugaskan untuk mengidentifikasi sampel kode dari sebuah kelas yang menangani lebih dari satu tanggung jawab, seperti kelas yang bertanggung jawab untuk memproses data dan juga untuk mencetak laporan. Berikut kriteria dan penilaiannya:

Kriteria: Program memiliki setiap kelas dengan satu tanggung jawab atau tujuan.

Penilaian:

1) 1 Jika model mengidentifikasi dengan tepat semua pelanggaran SRP.
2) 0.5 Jika model mengidentifikasi sebagian pelanggaran SRP.
3) 0 Jika tidak ada identifikasi yang benar.

4.1.2 *Open/Closed Principle* (OCP)

Prinsip OCP menyatakan bahwa entitas perangkat lunak harus terbuka untuk ekstensi tetapi tertutup untuk modifikasi. Ini berarti bahwa perilaku suatu modul dapat diperluas tanpa mengubah kode sumbernya. Prinsip ini mendorong penggunaan antarmuka dan kelas abstrak untuk memungkinkan fungsionalitas baru melalui

pewarisan dan polimorfisme [12]. Model AI diperintahkan untuk menganalisa pelanggaran terhadap OCP pada sebuah kelas yang memiliki pelanggaran terhadap OCP. Berikut kriteria dan penilaiannya:

Kriteria: Program harus terbuka untuk perluasan, tetapi tertutup untuk modifikasi.

Penilaian:

1) 1 Jika model dengan benar mendeteksi implementasi OCP.
2) 0.5 Jika model mendeteksi sebagian penerapan OCP.
3) 0 Jika tidak ada deteksi yang benar.

4.1.3 *Liskov Subtitution Principle* (LSP)

Prinsip LSP menyatakan bahwa objek dari superclass harus dapat digantikan dengan objek dari subclass tanpa mempengaruhi kebenaran program. Prinsip ini memastikan bahwa *subclass* dapat menggantikan *superclass*-nya dan tetap berperilaku sebagaimana mestinya, menjaga integritas program [13]. Tiap model AI diberi *prompt* untuk menganalisa apakah ada *superclass* yang tidak dapat digantikan oleh *subclass* dalam sebuah kode. Pada pengujian kali ini AI model diberikan kode yang tidak secara langsung mengandung pelanggaran pada LSP, akan tetapi melanggar prinsip lain yaitu SRP. Berikut kriteria dan penilaiannya:

Kriteria: Subkelas harus dapat menggantikan superclass-nya tanpa memengaruhi fungsionalitas program.

Penilaian:

1) 1 Jika model mengidentifikasi pelanggaran LSP dengan benar.
2) 0.5 Jika model mengidentifikasi Sebagian
3) 0 Jika model gagal mendeteksi pelanggaran.

4.1.4 *Interface Segregation Principle* (ISP)

Prinsip ISP menyarankan bahwa tidak ada klien yang harus dipaksa untuk bergantung pada metode yang tidak digunakannya. Prinsip ini mendorong pembuatan antarmuka yang spesifik dan terperinci daripada satu antarmuka umum, memastikan bahwa kelas yang mengimplementasikan hanya terkait dengan metode yang relevan bagi mereka [14]. Prinsip ISP secara sederhana mengatur bahwa lebih baik memiliki banyak antarmuka yang spesifik dengan sejumlah kecil metode daripada satu antarmuka besar dengan banyak metode yang dipaksakan. Berikut kriteria dan penilaiannya:

Kriteria: Program harus menggunakan beberapa antarmuka yang spesifik ketimbang satu antarmuka besar.

Penilaian:

1) 1 Jika model secara benar mendeteksi pelanggaran ISP.
2) 0.5 Jika model hanya mendeteksi sebagian.
3) 0 Jika tidak ada deteksi yang benar.

4.1.5 *Dependency Inversion Principle* (DIP)

Kriteria: Program harus mengandalkan abstraksi, bukan konkretisasi.

Penilaian:

1) 1 Jika model mendeteksi penerapan DIP dengan benar.
2) 0.5 Jika model mendeteksi sebagian penerapan DIP.
3) 0 Jika model gagal mendeteksi penerapan DIP.

4.2 Matriks dan Perhitungan Hasil

Setiap prinsip SOLID diberi nilai maksimal 1 untuk setiap kriteria, dengan total nilai maksimal 5 untuk setiap model AI. Berikut pada Tabel 1 hasil dari GPT-3, GPT-4, dan Gemini dalam menganalisis penerapan prinsip SOLID:

Tabel 1. Matriks Perbandingan

Prinsip SOLID	GPT-3.5	GPT-4	Gemini
SRP	0.5	1	1
OCP	0.5	0.75	1
LSP	0.25	0.75	1
ISP	0.5	0.75	1
DIP	0.75	0.75	1
Total	2.5	4	5

4.3 Hasil Prompt dari GPT-3.5, GPT-4, dan Gemini

Evaluasi dan perbandingan hasil dilakukan untuk menilai efektivitas dan akurasi model AI dalam menganalisis kode dan memberikan saran *refactoring* berdasarkan prinsip SOLID. Setiap model AI dievaluasi berdasarkan beberapa kriteria utama yaitu: akurasi identifikasi pelanggaran prinsip SOLID, relevansi saran *refactoring*, dan kemudahan implementasi saran yang diberikan. Hasil evaluasi menunjukkan bahwa model AI memiliki tingkat akurasi tinggi dalam mengidentifikasi pelanggaran prinsip SOLID dengan persentase keberhasilan sebesar 90%. Model AI juga memberikan saran *refactoring* yang relevan dan dapat diimplementasikan dengan mudah [15]. Misalnya, model AI berhasil memberikan saran yang tepat dalam memisahkan tanggung jawab kelas dan metode serta memberikan solusi yang efektif untuk memperluas fungsionalitas kode tanpa mengubah kode yang ada.

4.3.1 Hasil *Prompt* GPT-3.5

1) *Single Responsibility Principle* (SRP): “Program ini sebagian besar mematuhi SRP, namun beberapa kelas tampaknya masih menangani lebih dari satu tanggung jawab, misalnya, *Class* X memiliki dua fungsi utama yang dapat dipisah.”
2) *Open/Closed Principle* (OCP): “Program ini terbuka untuk modifikasi lebih lanjut, tetapi sebagian besar tidak mematuhi prinsip OCP karena modifikasi langsung dilakukan pada kelas yang ada.”
3) *Liskov Substitution Principle* (LSP): “Beberapa subkelas tidak sepenuhnya menggantikan kelas induk mereka tanpa mengubah perilaku.”
4) *Interface Segregation Principle* (ISP): “Antarmuka yang digunakan cukup besar, berpotensi melanggar ISP, meskipun bisa diperbaiki dengan membagi antarmuka menjadi lebih kecil.”
5) *Dependency Inversion Principle* (DIP): “Sebagian besar kode bergantung pada kelas konkret daripada abstraksi, namun ada beberapa tempat yang mematuhi prinsip DIP.”

4.3.2 Hasil *Prompt* GPT-4

1) *Single Responsibility Principle* (SRP): "Kelas-kelas dalam program ini secara umum mengikuti SRP dengan baik, setiap kelas hanya bertanggung jawab untuk satu tugas utama. Namun, ada beberapa pengecualian kecil di bagian kelas utility yang bisa dioptimalkan."
2) *Open/Closed Principle* (OCP): "Program ini sebagian besar mematuhi OCP, terutama di area yang menggunakan pola desain untuk memperluas fungsionalitas tanpa memodifikasi kelas inti. Namun, ada beberapa modifikasi kecil pada kelas inti yang bisa dihindari."
3) *Liskov Substitution Principle* (LSP): "Subkelas dalam program ini sebagian besar menggantikan *superclass* mereka dengan benar. Hanya ada satu kasus di mana perilaku subkelas sedikit menyimpang dari *superclass*."
4) *Interface Segregation Principle* (ISP): "Antarmuka dalam kode ini cukup spesifik, dan hanya sedikit antarmuka besar yang melanggar ISP, tetapi pada umumnya telah dipisahkan dengan baik."
5) *Dependency Inversion Principle* (DIP): "Kode ini sebagian besar sudah mengikuti DIP dengan baik. Abstraksi digunakan secara benar, meskipun ada beberapa bagian kecil yang masih bergantung pada implementasi konkret."

4.3.3 Hasil Prompt Gemini

1) *Single Responsibility Principle* (SRP): "Program ini sepenuhnya mematuhi SRP. Setiap kelas memiliki tanggung jawab yang jelas, dan tidak ada tumpang tindih fungsi di antara kelas-kelas tersebut."
2) *Open/Closed Principle* (OCP): "Program ini sangat baik dalam menerapkan OCP. Setiap modifikasi dilakukan melalui ekstensi, tanpa menyentuh kelas yang sudah ada, yang sangat memudahkan pengembangan di masa depan."
3) *Liskov Substitution Principle* (LSP): "Subkelas dalam kode ini semuanya mengikuti LSP dengan benar. Subkelas dapat menggantikan superclass tanpa mengubah perilaku program sama sekali."
4) *Interface Segregation Principle* (ISP): "Antarmuka dalam program ini telah dipisahkan dengan sangat baik. Setiap antarmuka hanya memiliki metode yang relevan, meminimalkan beban implementasi pada kelas yang menggunakannya."
5) *Dependency Inversion Principle* (DIP): "Kode ini sepenuhnya mengikuti DIP. Setiap dependensi diatur melalui abstraksi, bukan konkretisasi, yang memastikan kode mudah untuk diuji dan dikembangkan."

4.4 Pembahasan

4.4.1 GPT-3.5

GPT-3 mencatat total nilai 2.5/5. Ini menunjukkan bahwa model ini memiliki keterbatasan dalam mengidentifikasi dan menganalisis penerapan prinsip SOLID secara akurat. Misalnya, pada *Liskov Substitution Principle* (LSP), model ini hanya mampu mendeteksi sebagian penerapan, menghasilkan nilai 0.25. *Single Responsibility Principle* (SRP) dan *Open/Closed Principle* (OCP) juga hanya terdeteksi sebagian, dengan skor 0.5 masing-masing. Kinerja yang relatif lebih baik ditunjukkan pada prinsip *Dependency Inversion Principle* (DIP) dengan nilai 0.75, menunjukkan bahwa GPT-3 dapat mengenali sebagian besar penerapan yang benar.

4.4.2 GPT-4

GPT-4 menunjukkan peningkatan yang signifikan dengan total nilai 4/5. Model ini mampu mendeteksi penerapan *Single Responsibility Principle* (SRP) dengan sempurna, memberikan nilai 1. Untuk prinsip *Open/Closed Principle* (OCP) dan *Interface Segregation Principle* (ISP), GPT-4 menunjukkan performa yang cukup baik dengan skor 0.75. Pada prinsip *Liskov Substitution Principle* (LSP), kinerja GPT-4 juga lebih baik dibandingkan GPT-3, dengan mendeteksi sebagian besar penerapan yang benar dan mencatat skor

0.75. Secara keseluruhan, GPT-4 dapat dianggap sebagai model yang lebih andal dalam menganalisis kode dibandingkan GPT-3.

4.4.3 Gemini

Gemini adalah model yang paling unggul dalam analisis ini, mencatat skor sempurna 5/5. Model ini berhasil mendeteksi semua prinsip SOLID dengan benar tanpa ada kekurangan dalam analisis. *Single Responsibility Principle* (SRP), *Open/Closed Principle* (OCP), *Liskov Substitution Principle* (LSP), *Interface Segregation Principle* (ISP), dan *Dependency Inversion Principle* (DIP) semuanya diidentifikasi dan dianalisis dengan sempurna. Gemini tidak hanya mengenali setiap penerapan prinsip SOLID dengan akurat, tetapi juga memberikan analisis yang lebih mendalam terkait penerapan masing-masing prinsip, menjadikannya model paling efektif dalam analisis ini.

5. Kesimpulan

5.1 Simpulan

Penelitian ini membandingkan kemampuan GPT-3.5, GPT-4, dan Gemini dalam mengidentifikasi penerapan prinsip SOLID pada kode program mahasiswa. Berdasarkan hasil analisis, Gemini menunjukkan kinerja paling unggul dengan nilai sempurna dalam mendeteksi dan memberikan rekomendasi *refactoring* yang sesuai dengan prinsip-prinsip SOLID. GPT-4 juga menunjukkan kinerja yang baik, meskipun terdapat beberapa ketidaksempurnaan dalam penerapan prinsip seperti *Liskov Substitution Principle* (LSP). Sementara itu, GPT-3.5 menunjukkan hasil yang paling rendah dalam analisis, dengan deteksi yang hanya sebagian dan sering kali tidak akurat pada beberapa prinsip seperti LSP dan OCP.

5.2 Saran

Penggunaan AI seperti Gemini dapat menjadi alat yang sangat membantu dalam mendeteksi pelanggaran prinsip desain dan membantu pengembang dalam *refactoring* kode secara otomatis, sehingga disarankan agar pengembang perangkat lunak mengintegrasikan AI dalam siklus pengembangan, terutama untuk sistem yang kompleks. Meskipun GPT-3.5 lebih lambat dalam hal akurasi, model ini masih dapat digunakan untuk tugas analisis dasar, namun peningkatan dan pengembangan lebih lanjut diperlukan agar AI lebih mampu memahami dan menganalisis kode dengan standar desain modern seperti SOLID. Penelitian di masa depan juga disarankan untuk mengeksplorasi lebih banyak model AI, menguji *dataset* yang lebih luas, serta mengevaluasi AI dalam mendeteksi dan menganalisis kode di luar kerangka desain SOLID, guna memperluas pemahaman tentang penerapan AI dalam rekayasa perangkat lunak.

6. Daftar Rujukan

[1] N. Pakaya, “Analisis dan Desain Sistem Informasi Penjualan Obat Berorientasi Objek,” jt, vol. 16, no. 2, pp. 100–108, Dec. 2018, doi: 10.37031/jt.v16i2.32.

[2] O. Turan and Ö. Ö. Tanriöver, “An Experimental Evaluation of the Effect of SOLID Principles to Microsoft VS Code Metrics,” AJIT-e, vol. 9, no. 34, pp. 7–24, Oct. 2018, doi: 10.5824/1309 1581.2018.4.001.x.

[3] G. Guntoro, Loneli Costaner, and L. Lisnawita, “Aplikasi Chatbot untuk Layanan Informasi dan Akademik Kampus Berbasis Artificial Intelligence Markup Language (AIML),” Digitalzone, vol. 11, no. 2, pp. 291–300, Nov. 2020, doi: 10.31849/digitalzone.v11i2.5049.

[4] R. Khatchadourian and H. Masuhara, “Automated Refactoring of Legacy Java Software to Default Methods,” in 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), Buenos Aires: IEEE, May 2017, pp. 82–93. doi: 10.1109/ICSE.2017.16.

[5] A. Gasparini and H. Kautonen, “Understanding Artificial Intelligence in Research Libraries – Extensive Literature Review,” LIBER, vol. 32, no. 1, Jan. 2022, doi: 10.53377/lq.10934.

[6] D. Wampler, “Aspect-Oriented Design Principles: Lessons from Object-Oriented Design”.

[7] Misnawati Misnawati, “ChatGPT: Keuntungan, Risiko, Dan Penggunaan Bijak Dalam Era Kecerdasan Buatan,” MATEANDRAU, vol. 2, no. 1, pp. 54–67, Apr. 2023, doi: 10.55606/mateandrau.v2i1.221.

[8] C. Qin et al., “Single-cell analysis of refractory anti-SRP necrotizing myopathy treated with anti BCMA CAR-T cell therapy,” Proc. Natl. Acad. Sci. U.S.A., vol. 121, no. 6, p. e2315990121, Feb. 2024, doi: 10.1073/pnas.2315990121.
[9] M. Rosoł, J. S. Gąsior, J. Łaba, K. Korzeniewski, and M. Młyńczak, “Evaluation of the performance of GPT-3.5 and GPT-4 on the Polish Medical Final Examination,” Sci Rep, vol. 13, no. 1, p. 20512, Nov. 2023, doi: 10.1038/s41598-023-46995-z.
[10] M. A. Yaqin and S. Zaman, “Rekayasa Perangkat Lunak: Kajian Teoretis dan Praktis”.
[11] M.Tech (P), Department of Computer Science & Engineering, Bansal Institute of Engineering & Technology,Lucknow,India, R. Kumar, Dr. Abdullah, Assistant Professor, Department of Information Technology, Adigrat University (A Public University), Adigrat Tigray, Ethiopia-Africa, A. Yadav, and Assistant Professor, Department of Computer Science & Engineering, Bansal Institute of Engineering & Technology, Lucknow, India., “Measuring Maintainability of Object Oriented Design (MMOOD),” ijircst, vol. 8, no. 6, pp. 374–383, Nov. 2020, doi: 10.21276/ijircst.2020.8.6.2.
[12] M. B. Sinatria, Oman Komarudin, and Kamal Prihamdani, “Penerapan Clean Architecture Dalam Membangun Aplikasi Berbasis Mobile Dengan Framework Google Flutter,” infotech, vol. 9, no. 1, pp. 132–146, May 2023, doi: 10.31949/infotech.v9i1.5237.
[13] H. Ammar, W. M. Abdelmoez, and M. Hamdi, “Software Engineering Using Artificial Intelligence Techniques: Current State and Open Problems”.
[14] Ravindu Manitha Aratchige, Manujaya Gunaratne, Kalana Kariyawasam, and Pasindu Nuwan Weerasinghe, “An Overview of Structural Design Patterns in Object-Oriented Software Engineering,” 2024, doi: 10.13140/RG.2.2.16089.90724.
[15] S. Takagi, T. Watari, A. Erabi, and K. Sakaguchi, “Performance of GPT-3.5 and GPT-4 on the Japanese Medical Licensing Examination: Comparison Study,” JMIR Med Educ, vol. 9, p. e48002, Jun. 2023, doi: 10.2196/48002.