https://jurnal.pustakagalerimandiri.co.id/index.php/pustakadata
DOI: https://doi.org/10.55382/jurnalpustakadata.v6i3.1914

Jurnal Pustaka Data

Pusat Akses Kajian Database, Analisa Teknologi, dan Arsitektur Komputer

Vol . 6 No. 3 (2026) 219-224

E ISSN: 2809-7483

Rekonstruksi Diagram Kelas UML Moodle Menggunakan Analisis Kode Statis

Muktibaskara Kusbianto[1], Muhammad Aditya Dermawan[2], Muhammad Ainul Yaqin[3]
[1,2,3] Fakultas Sains dan Teknologi, Universitas Islam Negeri Maulana Malik Ibrahim Malang
[1]muktibaskrakusbianto@gmail.com,[2]adityadermawan111@gmail.com, [3]yaqinov@ti.uin-malang.ac.id

Abstract

Architectural design documentation in large-scale software such as Moodle is often inconsistent with its actual implementation, triggering technical debt. Consequently, automated extraction efforts frequently fail due to visual pollution resulting in "Spaghetti Diagrams". To address this specific issue, this study aims to resolve the visual pollution problem through a Static Code Analysis (SCA) approach based on the Abstract Syntax Tree (AST). This approach automatically reconstructs class diagrams across 220 Moodle source code files within the Assign, Course, and User modules. The evaluation is measured based on the quantity of successfully extracted architectural elements (classes, attributes, methods, relations) and the readability of the visual design. Extraction results indicate that the algorithm successfully processed all code without execution errors, with the Course Module recording the highest complexity level. The implementation of filtering and blacklisting mechanisms to discard utility classes and ignore local-level dependency relations proved crucial and effective in eliminating architectural noise. In conclusion, the reconstructed UML class diagram visualization is proven to be representative and accurate, serving as an actual architectural blueprint to facilitate continuous system maintenance.

Keywords: Reverse Engineering, Static Code Analysis, Abstract Syntax Tree, UML Class Diagram, Moodle

Abstrak

Dokumentasi desain arsitektur pada perangkat lunak berskala raksasa seperti Moodle sering kali tidak konsisten dengan implementasi aktualnya. Hal ini memicu penumpukan utang teknis (*technical debt*). Hal ini diperburuk dengan fakta bahwa upaya ekstraksi dokumentasi secara otomatis kerap gagal. Kegagalan ini terjadi karena menghasilkan "*Spaghetti Diagram*" akibat polusi visual dari detail implementasi tingkat rendah. Penelitian ini secara khusus bertujuan untuk menyelesaikan masalah polusi visual tersebut. Pendekatan yang digunakan adalah Analisis Kode Statis (*Static Code Analysis* / SCA) berbasis *Abstract Syntax Tree* (AST). Pendekatan ini digunakan untuk merekonstruksi diagram kelas secara otomatis pada 220 *file* kode sumber Moodle di modul Assign, Course, dan User. Evaluasi diukur berdasarkan kuantitas elemen arsitektur (kelas, atribut, metode, relasi) yang berhasil diekstraksi serta keterbacaan desain visualnya. Hasil penelusuran menunjukkan algoritma berhasil memproses seluruh kode tanpa galat. Modul Course mencatat tingkat kompleksitas tertinggi yang didominasi oleh relasi pewarisan (*inheritance*) dan agregasi (*aggregation*). Implementasi mekanisme *filtering* dan *blacklisting* untuk membuang kelas utilitas terbukti krusial. Selain itu, mengabaikan relasi dependensi (*dependency*) tingkat lokal juga terbukti efektif dalam mengeliminasi *architectural noise*. Kesimpulannya, visualisasi diagram kelas UML yang direkonstruksi terbukti representatif dan akurat. Diagram ini dapat digunakan sebagai *blueprint* arsitektur yang aktual guna memfasilitasi pemeliharaan sistem lanjutan.

Kata kunci: Reverse Engineering, Analisis Kode Statis, Abstract Syntax Tree, Diagram Kelas UML, Moodle

Submitted : 29-05-2026 | Reviewed : 22-06-2026 | Accepted : 22-06-2026

1. Pendahuluan

Pada perangkat lunak sumber terbuka yang telah berkembang lama seperti *Learning Management System* (LMS) Moodle, dokumentasi desain arsitektur sering kali tidak diperbarui secara konsisten. Moodle merupakan *platform* raksasa dengan basis kode yang sangat masif dan terus berevolusi. Kondisi ini menyebabkan ketidaksesuaian antara desain konseptual dengan implementasi aktual sistem. Padahal, keselarasan antara implementasi aktual sistem dengan analisis kebutuhan dan desain awal sangatlah krusial untuk menentukan efektivitas serta kualitas perangkat lunak (Permatasari dkk., 2025). Jika dibiarkan, hal ini akan memicu penumpukan utang teknis (*technical debt*) yang membuat sistem sangat sulit untuk dipelihara. Sebagai solusinya, rekayasa balik (*reverse engineering*) menjadi pendekatan krusial dalam pemulihan arsitektur perangkat lunak. Tujuannya adalah mengidentifikasi komponen sistem dan merepresentasikannya pada tingkat abstraksi yang lebih tinggi (Chikofsky & Cross, 1990).

Dalam perkembangannya, berbagai penelitian terdahulu sangat mengandalkan teknik Analisis Kode Statis (*Static Code Analysis* / SCA). Teknik ini digunakan untuk mengekstraksi informasi arsitektural langsung dari kode sumber tanpa perlu mengeksekusi program secara dinamis (Al-Msie'deen, 2025). Pendekatan SCA memungkinkan ekstraksi otomatis struktur kode fundamental seperti kelas, atribut, metode, dan relasi antarobjek. Elemen ini kemudian ditransformasikan menjadi representasi visual seperti diagram kelas UML (Khalid & Ibrahim, 2016). Penggunaan SCA dinilai sangat efisien untuk memfasilitasi pemahaman program dan rekonstruksi arsitektur pada sistem warisan (*legacy systems*). Pada sistem ini, dokumentasi desainnya sering kali hilang atau tidak lagi selaras dengan implementasi kode aktual (Kaliappan & Ali, 2018).

Secara teknis, mayoritas studi terdahulu melakukan analisis statis dengan memanfaatkan teknologi *parser*. Teknologi ini menghasilkan representasi *Abstract Syntax Tree* (AST) sebagai langkah awal mengekstraksi informasi semantik dan struktural dari kode program (Zhang, 2016; Acharya, 2013). Setelah pohon sintaksis AST terbentuk, algoritma penelusuran diterapkan untuk memetakan entitas kode ke dalam elemen visual UML. Pemetaan ini didasarkan pada aturan pemetaan (*mapping rules*) spesifik yang berlaku pada pemrograman berorientasi objek (Jindal dkk., 2008). Dalam konteks penelitian ini, instrumen analisis dibangun menggunakan *library* nikic/php-parser untuk memetakan arsitektur kelas secara presisi. Meskipun penelitian sebelumnya telah membuktikan keberhasilan otomatisasi ekstraksi arsitektur, sebagian besar instrumen didominasi oleh pemrosesan bahasa kompilasi seperti Java dan C++. Instrumen tersebut juga cenderung diuji pada perangkat lunak berskala kecil hingga menengah (Sutton & Maletic, 2005; Sharma & Chandel, 2012).

Meskipun pendekatan ekstraksi otomatis memberikan banyak kemudahan, terdapat keterbatasan signifikan yang menjadi tantangan utama. Alat bantu ekstraksi arsitektur tradisional sering kali gagal memberikan abstraksi desain yang memadai. Alat ini justru mereproduksi diagram yang terlalu *granular* dan sarat akan *architectural noise* (Hatahet dkk., 2025; Sutton & Maletic, 2005). *Architectural noise* merupakan polusi visual berupa elemen detail implementasi tingkat rendah seperti utilitas generik. Elemen ini mengaburkan dan tidak relevan dengan pemahaman arsitektur makro suatu sistem. Akibatnya, upaya visualisasi keseluruhan pada sistem berskala raksasa tanpa alat bantu penyaringan justru membebani pengamat (Talerico, 2003). Ekstraksi seluruh relasi antarentitas dilakukan tanpa mekanisme penyaringan yang terarah. Hal ini mencakup dependensi lokal antar-metode dan pemanfaatan kelas utilitas yang pada akhirnya menciptakan polusi visual tumpang tindih. Tampilan visual yang sangat rumit dan ketiadaan struktur yang jelas ini lazim dikenal sebagai *spaghetti diagram*. Keterbatasan visual inilah yang menjadi celah penelitian (*research gap*) dalam ranah pemulihan arsitektur pada penelitian ini.

Sangat sedikit studi empiris yang berfokus pada rekonstruksi arsitektur sistem berbasis bahasa skrip PHP. Terlebih lagi untuk audit struktural langsung pada sistem *enterprise* dengan basis kode masif seperti LMS Moodle. Oleh karena itu, penelitian ini mengusulkan pendekatan analisis statis berbasis AST yang mengintegrasikan strategi resolusi polusi visual. Strategi ini meliputi mekanisme *blacklisting* terhadap kelas utilitas generik dan teknik *filtering* untuk mengabaikan relasi dependensi tingkat lokal. Implementasi strategi ini sangat krusial guna mengeliminasi *architectural noise* secara selektif. Hal ini memastikan diagram kelas UML yang direkonstruksi menjadi lebih bersih, memiliki keterbacaan tinggi, dan berfungsi sebagai *blueprint* aktual pemeliharaan sistem. Oleh karena itu, penelitian ini bertujuan menjawab rumusan masalah: Bagaimana mengurangi polusi visual pada diagram hasil *reverse engineering* Moodle?

Submitted : 29-05-2026 | Reviewed : 22-06-2026 | Accepted : 22-06-2026

2. Metode Penelitian

2.1 Desain Eksperimen

Penelitian ini menggunakan pendekatan *Static Code Analysis* (SCA) untuk melakukan rekayasa balik (*reverse engineering*) terhadap arsitektur perangkat lunak. Eksperimen dirancang untuk mengevaluasi kemampuan algoritma analisis statis. Tujuannya adalah merekonstruksi diagram kelas secara akurat dan representatif dari basis kode raksasa Moodle. Variabel bebas yang ditetapkan adalah tiga modul utama Moodle dengan karakteristik berbeda. Modul tersebut yaitu Modul Aktivitas Pembelajaran (Assign), Modul Manajemen Kursus (Course), dan Modul Manajemen Pengguna (User). Variabel terikat yang diukur adalah kuantitas dan kelengkapan elemen arsitektural (kelas, atribut, metode, relasi) yang berhasil diekstraksi. Selain itu, tingkat keterbacaan visual setelah mekanisme penyaringan (*filtering*) diterapkan juga diukur.

2.2 Karakteristik Data dan Kode Sumber

Data primer dalam penelitian ini bersumber dari *file* kode sumber berbahasa PHP milik sistem LMS Moodle versi 5.1.3. Basis kode Moodle memiliki karakteristik berorientasi objek yang sangat padat. Satu *file* umumnya merepresentasikan satu entitas kelas utama dengan berbagai relasi dependensi, pewarisan (*inheritance*), dan agregasi (*aggregation*). Untuk memberikan gambaran konkret mengenai kompleksitas struktural data, berikut adalah cuplikan (*snippet*) dari *file* course_request.php di Modul Course:

```PHP
namespace core_course;

use core\{clock, di};
use core\context\course as context_course;
use stdClass;

class course_request {
    protected $properties;
    protected static $summaryeditoroptions;

    public static function prepare($data = null) { ... }

    public static function create($data) { ... }

    public function __construct($properties) { ... }

    public function get_category() { ... }

    public function approve() { ... }

    public function reject($notice) { ... }
}
```

Potongan kode di atas menunjukkan struktur dasar kelas course_request yang memiliki properti privat/*protected* dan kumpulan metode fungsional. Elemen non-krusial dari implementasi tingkat rendah akan diabaikan pada tahap pemindaian. Fokus utama pemindaian adalah mengekstraksi deklarasi kelas, atribut, metode, dan injeksi dependensi (*dependency injection*).

2.3 Tahapan Analisis Berbasis Abstract Syntax Tree (AST)

Proses rekayasa balik diawali dengan pembentukan *Abstract Syntax Tree* (AST) menggunakan pustaka komponen *parser* pihak ketiga, yaitu *library* nikic/php-parser versi 5.x. Versi ini dipilih karena memberikan dukungan penuh terhadap pemrosesan sintaksis modern pada *runtime* PHP 8.x yang digunakan oleh Moodle versi 5.1.3. Kode sumber PHP akan dipindai menjadi susunan token (*tokenizing*), diverifikasi tata bahasanya (*grammatical verification*), lalu direpresentasikan ke dalam struktur pohon sintaksis di dalam memori.

Setelah struktur AST terbentuk, algoritma *Visitor Pattern* yang diimplementasikan pada kelas MoodleUmlVisitor akan menelusuri setiap simpul (*node*) pohon secara rekursif untuk mengekstraksi metrik arsitektural. Berikut ini adalah pseudocode dari algoritma *traversal* AST yang dirancang:

Berikut ini adalah pseudocode dari algoritma traversal AST yang dirancang:

Algoritma 1: Ekstraksi Fakta Arsitektural Berbasis AST

```PHP
INPUT : dir_moodle
OUTPUT: uml_data

1: init Parser, Visitor
2: foreach file in dir_moodle:
3:    ast = Parser.parse(file.read())
4:    foreach node in ast:
5:       if node is Class | Interface:
6:          uml_data.add(node.name)
7:          if node.extends:
8:             add_relation(Inheritance)
9:          foreach stmt in node.body:
10:            if stmt is Property:
11:               add_attribute(stmt)
12:            if stmt is Method:
13:               add_operation(stmt)
14:               if stmt.name == '__construct':
15:                  extract_params() -> Aggregation
16:                  extract_new() -> Composition
17: return uml_data
```

Narasi Pemrosesan Kode:

Ketika Algoritma 1 memproses cuplikan course_request.php dari Sub-bab 2.2, *Parser* dari nikic/php-parser v5.x akan mengenali class course_request sebagai *Node ClassDeclaration* (Baris 5). Algoritma kemudian masuk ke dalam *body class* (Baris 9) dan mendeteksi $properties serta $summaryeditoroptions sebagai *Node Property* (Baris 10), mencatatnya sebagai atribut kelas dengan visibilitas *protected* (#). Selanjutnya, fungsi seperti approve() dan __construct() dikenali sebagai *Node Method* (Baris 12). Khusus pada metode konstruktor (__construct), parameter yang disuntikkan (*dependency injection*) akan dipindai lebih dalam untuk mendeteksi relasi *Aggregation* ke kelas lain. Data ini dikumpulkan secara komprehensif ke dalam

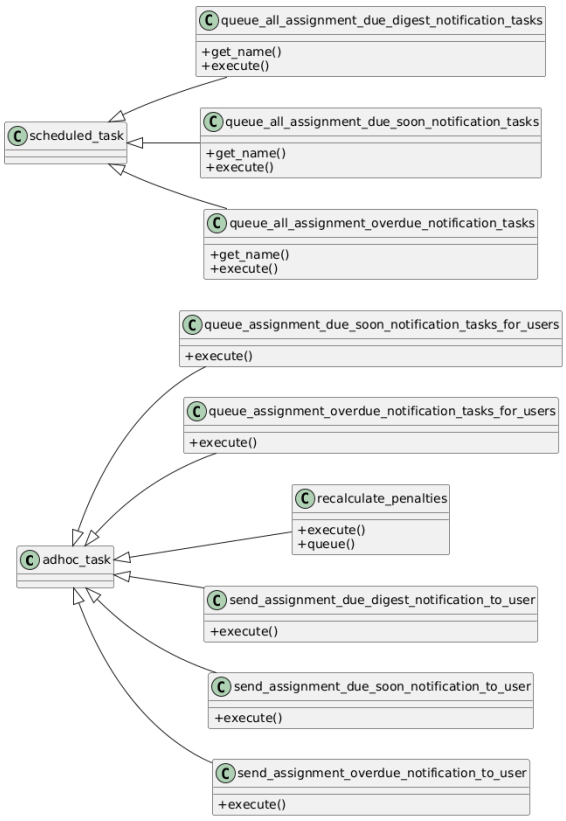
memori (*associative array*) sebelum di-*render* menjadi visualisasi UML.

2.4 Aturan Pemetaan (Mapping Rules)

Proses transformasi dari kode sumber PHP menjadi diagram kelas UML didasarkan pada serangkaian aturan pemetaan struktural. Pemetaan ini memverifikasi *node-node* AST dan mengubahnya menjadi elemen visual dengan notasi standar UML. Berikut ini adalah aturan pemetaan yang diimplementasikan:

Tabel 2.4. Aturan Pemetaan (Mapping Rules) Struktur AST ke Notasi UML

Struktur Kode PHP	Elemen Diagram UML	Simbol / Notasi
class [Nama]	Class	Kotak Kelas
interface [Nama]	Interface	<<interface>>
public $property	Public Attribute	+ namaAtribut
protected $property	Protected Attribute	# namaAtribut
private $property	Private Attribute	- namaAtribut
public function name()	Public Operation	+ namaMetode()
extends [Parent]	Inheritance	Garis lurus, panah kosong (solid)
implements [Interface]	Realization	Garis putus-putus, panah kosong
new [Class] / Params	Aggregation / Association	Garis dengan ujung berlian/panah



Gambar 2.5 Cuplikan Diagram Kelas UML pada Modul Assign

Sebagai ilustrasi penerapan aturan pemetaan, Gambar 2.5 menampilkan cuplikan kecil diagram kelas hasil ekstraksi pada Modul Aktivitas Pembelajaran (Assign). Pada cuplikan ini, mesin *parser* secara presisi mendeteksi relasi pewarisan (*inheritance*) antara kelas induk seperti adhoc_task dan scheduled_task dengan berbagai kelas turunannya. Contoh turunannya adalah recalculate_penalties, send_assignment_due_soon_notification_to_user, dan queue_all_assignment_due_soon_notification_tasks.

2.5 Strategi Resolusi Architectural Noise.

Dalam pemrosesan perangkat lunak berskala besar, ekstraksi otomatis tanpa *filter* akan berujung pada representasi visual yang sangat padat dan tidak dapat dipahami. Hal ini memunculkan urgensi untuk mereduksi *noise* dari desain. *Architectural Noise* dalam konteks rekayasa balik (*reverse engineering*) perangkat lunak adalah polusi visual berupa tumpukan entitas *class* utilitas generik dan relasi *dependency* tingkat lokal yang terlalu detail (*granular*). Keberadaan elemen-elemen ini tidak memberikan kontribusi signifikan terhadap pemahaman arsitektur tingkat tinggi (abstraksi konseptual), melainkan justru mendegradasi keterbacaan visual hingga membentuk *Spaghetti Diagram*.

Sebagai contoh, di dalam metode prepare() pada *file* course_request.php seringkali ditemukan inisiasi objek utilitas standar bawaan PHP seperti berikut:

```
PHP
    public static function prepare($data = null) {
        if ($data === null) {
            $data = new stdClass(); // Memicu polusi
    dependensi lokal
        }
        // ...
        return $data;
    }
```

Jika inisiasi kelas stdClass atau kelas penanganan *error* seperti Exception diekstraksi ke diagram, sistem akan menarik ratusan garis dependensi. Garis ini ditarik dari hampir semua kelas Moodle menuju satu kelas utilitas, menciptakan jaring laba-laba visual yang merusak desain arsitektur utama. Selain itu, implementasi masif pada antarmuka generik sistem (seperti renderable, templatable, dan moodleform) juga memicu penumpukan relasi *realization* yang mengaburkan desain.

Untuk mencegah hal ini, penelitian ini mengimplementasikan algoritma penyaringan (*filtering*) dan pendaftaran hitam (*blacklisting*) yang terintegrasi pada fase PlantUmlBuilder

Algoritma 2: Resolusi Polusi Visual (*Filtering & Blacklisting*)

```
PHP
    INPUT : uml_data, blacklist
    OUTPUT: puml_script
```

```
1: init puml_script
2: foreach class_name, data in uml_data:
3:    if class_name in blacklist:
4:       continue
5:
6:    puml_script.append(data.structure)
7:
8:    foreach rel_type, targets in data.relations:
9:       if rel_type == 'dependency':
10:         continue
11:
12:      foreach target in targets:
13:         if target not in blacklist:
14:            puml_script.append(render_uml_arrow())
15: return puml_script
```

Melalui implementasi Algoritma 2 ini, setiap kelas yang teridentifikasi dalam *blacklist* (seperti stdClass) tidak akan dirender blok kelas maupun garis relasinya. Begitu pula dengan relasi *dependency* lokal yang terbentuk di dalam badan metode biasa. Relasi ini sengaja diabaikan untuk menjaga abstraksi tetap pada tingkatan arsitektur struktural (hanya berfokus pada *Inheritance*, *Realization*, *Aggregation*, dan *Composition*).

3. Hasil dan Pembahasan

3.1 Analisis Kuantitatif

Hasil ekstraksi kuantitatif terhadap ketiga modul Moodle secara rinci disajikan pada Tabel 3.1. Data tersebut mencakup distribusi jumlah kelas, *interface*, atribut, serta metode yang berhasil dipetakan oleh mesin *parser* dari keseluruhan *file* kode sumber.

Tabel 3.1. Ringkasan Hasil Ekstraksi Metrik Arsitektur Moodle

Modul Moodle	Total File PHP	Total Kelas	Total Interface	Total Atribut	Total Metode
Assign	84	78	4	110	398
Course	96	92	2	182	691
User)	40	40	0	63	214

Berdasarkan metrik pada Tabel 3.1, dapat diamati bahwa Modul Course memiliki tingkat kompleksitas arsitektural tertinggi dibandingkan modul lainnya, yang ditandai dengan ekstraksi 691 metode dari 96 file kode sumber. Tingginya volume atribut dan metode pada modul ini merepresentasikan besarnya tanggung jawab entitas course sebagai pusat kendali utama dari aktivitas pembelajaran di dalam sistem Moodle.

3.2 Analisis Kualitatif dan Perbandingan

Pada tahap ini, evaluasi dilakukan terhadap kejelasan struktur diagram, keterbacaan desain, serta kesesuaian relasi dengan implementasi kode. Berdasarkan *output* visual dari sistem PlantUML, diagram kelas UML berhasil dibangun menggunakan notasi standar berdasarkan pemetaan struktur kode. Penggunaan notasi standar ini sekaligus berperan untuk menjamin validitas konstruk dalam penelitian. Melalui implementasi aturan ekstraksi yang konsisten, validitas internal sistem juga dapat dijaga dengan baik. Sementara itu, validitas eksternal penelitian dibahas melalui potensi penerapan metode yang sama. Metode ini menggunakan mesin *parser* AST dan *filter* relasi untuk mengekstraksi arsitektur pada sistem berbasis objek lainnya.

Terkait dengan analisis perbandingan, prosedur idealnya adalah membandingkan diagram hasil rekonstruksi dengan dokumentasi desain resmi Moodle untuk menilai kesesuaian. Namun, pada perangkat lunak sumber terbuka yang berkembang lama, dokumentasi desain sering kali tidak diperbarui konsisten. Kondisi tersebut memicu terjadinya ketidaksesuaian antara desain konseptual awal dengan implementasi aktual saat ini. Oleh karena itu, perbandingan dititikberatkan pada keberhasilan diagram hasil *reverse engineering* dalam menyajikan *blueprint* arsitektur yang aktual.

Diagram kelas yang dihasilkan terbukti akurat dalam memetakan struktur internal sistem Moodle secara aktual. Diagram ini juga bersih dari tumpukan relasi *dependency* tingkat lokal yang merusak keterbacaan desain. Dengan demikian, visualisasi ini dapat diandalkan sebagai dasar pemahaman sistem, pemeliharaan, dan pengembangan lanjutan, menggantikan dokumentasi usang. Sebagai contoh konkret, *developer* dapat melihat bahwa pembaruan pada satu kelas tertentu (misal: komponen grade) dapat dilakukan tanpa merusak puluhan kelas lain.

Tabel 3.2. Distribusi Jenis Relasi per Modul

Modul Moodle	Inheritance	Realization	Association	Aggregation	Composition	Total Relasi
Assign	60	22	2	9	1	94
Course	46	40	3	36	2	127
User	24	14	0	6	0	44

Sebagai catatan, metrik kuantitatif pada Tabel 3.1 dan 3.2 merupakan total elemen arsitektur yang diekstraksi mentah oleh mesin *parser*. Pada hasil visualisasi akhir PlantUML, beberapa antarmuka UI dan utilitas generik sengaja disaring melalui *blacklisting* untuk menjaga kebersihan abstraksi visual.

Selanjutnya, merujuk pada distribusi relasi di Tabel 3.2, terlihat jelas bahwa relasi *Inheritance* (pewarisan) dan *Aggregation* mendominasi struktur desain ketiga modul tersebut. Tingginya angka *Inheritance* membuktikan bahwa *framework* Moodle sangat mengandalkan prinsip *extensibility*, di mana fungsionalitas baru mewarisi *core classes*. Hal ini disebabkan karena arsitektur Moodle banyak menggunakan desain pola *Template Method*. Pada pola ini, kerangka kerja utama didefinisikan pada *superclass* statis dan detail implementasinya diserahkan pada *subclass* turunan. Temuan ini memvalidasi bahwa pendekatan analisis statis mampu menangkap karakteristik *Object-Oriented* dari sistem

warisan secara komprehensif. Untuk melihat visualisasi penuh dari diagram kelas UML hasil rekonstruksi, tautan resolusi tinggi dapat diakses melalui repositori GitHub di lampiran. berikut: [https://github.com/mug31/moodle-analyzer.git].

3.3 Batasan Penelitian

Batasan dari penelitian ini adalah analisis masih difokuskan secara eksklusif pada arsitektur struktural statis Moodle (PHP). Penelitian ini belum menguji generalisasi alat pada bahasa pemrograman lain. Selain itu, analisis statis memiliki kelemahan inheren yaitu tidak dapat menangkap relasi dinamis atau polimorfisme (*polymorphism*) yang terbentuk saat *runtime*. Oleh karena itu, penelitian belum mencakup analisis kode dinamis (*dynamic analysis*) untuk mengevaluasi perilaku operasional sistem. Ruang lingkup penelitian ini juga hanya mencakup diagram kelas dan belum mengekspansi ke *package diagram* atau *sequence diagram*. Pengembangan selanjutnya dapat mengeksplorasi skalabilitas algoritma untuk sistem dengan spesifikasi yang lebih besar serta penggabungan analisis dinamis.

4. Kesimpulan

Berdasarkan hasil eksperimen dan pembahasan, dapat ditarik kesimpulan sebagai berikut:

1. Pendekatan Analisis Kode Statis (SCA) berbasis *Abstract Syntax Tree* (AST) terbukti sangat handal mengekstraksi fakta arsitektural sistem berskala raksasa. Mesin *parser* berhasil memindai dan memetakan elemen berorientasi objek dari 220 *file* kode sumber PHP (Modul Assign, Course, dan User) tanpa galat sintaksis.
2. Implementasi mekanisme *filtering* dan *blacklisting* (penghapusan kelas utilitas dan antarmuka generik Moodle) terbukti krusial dan efektif mengeliminasi *architectural noise*.
3. Keputusan untuk mengabaikan relasi *Dependency* yang bersifat lokal pada tingkat metode berhasil mencegah terbentuknya *Spaghetti Diagram*. Diagram kelas UML yang direkonstruksi menjadi jauh lebih bersih, memiliki keterbacaan tinggi, dan murni menonjolkan relasi struktural utama.
4. Hasil rekonstruksi visual ini memecahkan masalah utang teknis (*technical debt*) terkait dokumentasi usang, memberikan *blueprint* desain sinkron untuk pemeliharaan sistem lanjutan.

Daftar Rujukan

[1] Acharya, R. (2013). "Object Oriented Design Pattern Extraction From Java Source Code," *Master's Thesis*, Uppsala Universitet, Swedia.
[2] Budhkar, S., and Gopal, A. (2012). "Component-Based Architecture Recovery from Object Oriented Systems using Existing Dependencies among Classes," *International Journal of Computational Intelligence Techniques*, vol. 3, no. 1.
[3] Chikofsky, E. J., and Cross, J. H. (1990). "Reverse Engineering and Design Recovery: A Taxonomy," *IEEE Software*, vol. 7, no. 1, pp. 13–17.
[4] Favre, L. (2010). *Model Driven Architecture for Reverse Engineering Technologies: Strategic Directions and System Evolution*. Hershey, PA, USA: Engineering Science Reference (IGI Global).
[5] Jindal, V., Jain, A., and Tayal, D. K. (2008). "On reverse engineering an object-oriented code into UML class diagrams incorporating extensible mechanisms," *ACM SIGSOFT Software Engineering Notes*, vol. 33, no. 5.
[6] Kaliappan, V., and Ali, N. M. (2018). "Improving Consistency of UML Diagrams and Its Implementation Using Reverse Engineering Approach," *Bulletin of Electrical Engineering and Informatics (BEEI)*, vol. 7, no. 4.
[7] Khalid, S., and Ibrahim, R. (2016). "Generating UML class diagram from source codes using multi-threading technique," *ARPN Journal of Engineering and Applied Sciences*, vol. 11, no. 12.
[8] Mohamed, K. A., and Kamel, A. (2018). "Reverse Engineering State and Strategy Design Patterns using Static Code Analysis," *The Science and Information (SAI) Organization*, vol. 9, no. 1.
[9] Rajput, A. (2014). "Reverse Engineering: Java code to uml diagram showing dependencies," *Bachelor of Technology Project*, Jaypee University of Information Technology, India.
[10] Rasool, G., and Asif, N. (2007). "Software Architecture Recovery," *World Academy of Science, Engineering and Technology*, vol. 34.
[11] Richner, T., and Ducasse, S. (1999). "Recovering High-Level Views of Object-Oriented Applications from Static and Dynamic Information," *Proceedings of the International Conference on Software Maintenance (ICSM'99)*, IEEE.
[12] Schneider, S., Bakhtin, A., Li, X., Soldani, J., Brogi, A., Cerny, T., Scandariato, R., and Taibi, D. (2024). "Comparison of Static Analysis Architecture Recovery Tools for Microservice Applications," *Proceedings of Mining Software Repositories (MSR'24)*, ACM.
[13] Sharma, N., and Chandel, G. S. (2012). "Generates UML Diagrams From Java Code Through Reverse Engineering," *International Journal of Engineering and Computer Science (IJECCE)*, vol. 3, no. 3.
[14] Sutton, A., and Maletic, J. I. (2005). "Mappings for Accurately Reverse Engineering UML Class Models from C++," *Proceedings of the 12th Working Conference on Reverse Engineering (WCRE'05)*, IEEE.
[15] Talerico, D. (2003). "Grouping in Object-Oriented Reverse Engineering," *Master's Thesis*, Universität Bern, Swiss.
[16] Tonella, P., and Potrich, A. (2002). "Static and Dynamic C++ Code Analysis for the Recovery of the Object Diagram," *Proceedings of the International Conference on Software Maintenance*, IEEE.
[17] Zhang, H. (2016). "An Approach for Extracting UML Diagram from Object-Oriented Program Based on J2X," *International Forum on Mechanical, Control and Automation (IFMCA)*, Atlantis Press.
[18] A. D. Permatasari, N. Rahmatina, and M. A. Yaqin, "Evaluasi Teknik Elisitasi pada Software Requirement dalam Menentukan Efektivitas Kebutuhan Perangkat Lunak," *Jurnal Pustaka Data (Pusat Akses Kajian Database, Analisa Teknologi, dan Arsitektur Komputer)*, 2025.

Submitted : 19-05-2026 | Reviewed : 26-05-2026 | Accepted : 31-05-2026