https://jurnal.pustakagalerimandiri.co.id/index.php/pustakadata
DOI: https://doi.org/10.55382/jurnalpustakadata.v6i3.1959

Jurnal Pustaka Data

Pusat Akses Kajian Database, Analisa Teknologi, dan Arsitektur Komputer

Vol. 6 No. 3 (2026) 359 – 365

E ISSN: 2809-7483

Studi Dampak Refactoring terhadap Kompleksitas Kode Moodle

Muhammad Haikal Fikri[1], Ahmad Daffa Febrian[2], Muhammad Ainul Yaqin[3]
Teknik Informatika, Sains dan Teknologi, Universitas Islam Negeri Maulana Malik Ibrahim Malang
[1]230605110067@student.uin-malang.ac.id, [2]230605110068@student.uin-malang.ac.id, [3]yaqinov@ti.uin-malang.ac.id

Abstract

Large-scale learning management systems like Moodle possess highly complex code structures. Refactoring is a widely used technique to improve the internal structure of code without changing its external behavior to address such complexity issues. This study aims to analyze the quantitative impact of implementing Extract Method and Pull Up Method techniques on code complexity within the Moodle version 5.1.2 assignment module. The research methodology involves static analysis using the PDepend tool to measure Weighted Methods per Class (WMC) and Lines of Code (LOC) at the class level, alongside Cyclomatic Complexity Number (CCN) at the method level across five target event classes. The results demonstrate that the Pull Up Method successfully reduced both WMC and LOC in the feedback_viewed and grading_form_viewed classes, while decreasing the CCN of their validation methods by 33% (from 3 to 2). Conversely, the Extract Method applied to assessable_submitted reduced the CCN of its primary method by 66% (from 3 to 1). Despite introducing structural overhead that increased class-level LOC by up to 14% and raised WMC due to the declaration of new helper methods, the logic became more linear. Meanwhile, the base class centralized shared validation complexity, increasing its WMC from 10 to 12. This study concludes that while refactoring introduces structural overhead, it successfully mitigates method-level logical complexity, thereby enhancing code modularity and long-term maintainability.

Keywords: Refactoring, Moodle, Code Complexity, WMC, CCN, PDepend.

Abstrak

Sistem manajemen pembelajaran berskala besar seperti Moodle memiliki struktur kode yang sangat kompleks. Refactoring merupakan teknik yang digunakan secara luas untuk memperbaiki struktur internal kode tanpa mengubah perilaku eksternalnya guna mengatasi masalah kompleksitas tersebut. Penelitian ini bertujuan untuk menganalisis dampak kuantitatif dari penerapan teknik Extract Method dan Pull Up Method terhadap kompleksitas kode di dalam modul assignment Moodle versi 5.1.2. Metodologi penelitian melibatkan analisis statis menggunakan alat PDepend untuk mengukur Weighted Methods per Class (WMC) dan Lines of Code (LOC) pada tingkat kelas, serta Cyclomatic Complexity Number (CCN) pada tingkat metode terhadap lima kelas event target. Hasil penelitian menunjukkan bahwa teknik Pull Up Method berhasil menurunkan nilai WMC dan LOC pada kelas feedback_viewed dan grading_form_viewed, sekaligus mengurangi nilai CCN pada metode validasinya sebesar 33% (dari 3 menjadi 2). Sebaliknya, teknik Extract Method yang diterapkan pada kelas assessable_submitted menurunkan nilai CCN metode utamanya sebesar 66% (dari 3 menjadi 1). Meskipun memperkenalkan overhead struktural yang meningkatkan LOC tingkat kelas hingga 14% dan menaikkan WMC akibat deklarasi metode pembantu baru, alur logika program menjadi jauh lebih linear. Sementara itu, kelas base menyentralisasikan kompleksitas validasi bersama, sehingga meningkatkan nilai WMC kelas tersebut dari 10 menjadi 12. Penelitian ini menyimpulkan bahwa walaupun refactoring memperkenalkan overhead struktural, teknik ini sukses memitigasi kompleksitas logis pada tingkat metode, sehingga meningkatkan modularitas kode dan keterpeliharaan jangka panjang.

Kata kunci: Refactoring, Moodle, Kompleksitas Kode, WMC, CCN, PDepend.

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

1. Pendahuluan

Sistem manajemen pembelajaran berskala besar seperti Moodle memiliki arsitektur yang sangat luas dan kompleks. Seiring berjalannya waktu, sistem ini terus mengalami evolusi berkelanjutan untuk memenuhi tuntutan fungsionalitas dan kebutuhan pengguna yang dinamis, namun siklus evolusi tersebut sering kali memicu penurunan kualitas kode atau degradasi struktur internal perangkat lunak [11], [15]. Berdasarkan standar rekayasa perangkat lunak, kompleksitas yang tidak terkelola dengan baik pada sistem informasi dapat meningkatkan beban kerja pengembang dalam melakukan pemeliharaan, memperparah akumulasi *technical debt* berupa ketidakefisienan desain, serta mempersulit proses identifikasi dan perbaikan kesalahan [6], [12], [13]. Selain itu, munculnya cacat atau kesalahan logika pada kode sumber yang tidak terdeteksi berpotensi meningkatkan biaya perawatan perangkat lunak, baik dalam jangka pendek maupun jangka panjang [10]. Oleh karena itu, diperlukan upaya sistematis untuk menjaga kualitas perangkat lunak agar tetap optimal selama siklus hidupnya [8], salah satunya melalui pendeteksian kesalahan sedini mungkin dengan memanfaatkan pemantauan instrumen metrik kompleksitas secara statis [10].

Salah satu teknik yang terbukti paling efektif dalam menangani permasalahan degradasi kode dan mengeliminasi duplikasi segmen kode adalah refactoring [13]. Refactoring merupakan proses mengubah struktur internal perangkat lunak untuk membuatnya lebih mudah dipahami, lebih modular, meningkatkan ekstensibilitas, dan lebih murah untuk dimodifikasi tanpa mengubah perilaku eksternalnya [11], [13]. Teknik ini sangat krusial dalam meningkatkan kualitas keterpeliharaan dan efisiensi pada sistem informasi yang memiliki ketergantungan data yang tinggi [3], [6], [15]. Penelitian terkini telah mengeksplorasi berbagai implementasi *refactoring*, mulai dari peningkatan kualitas sistem informasi kesehatan [3], [4], [6], optimalisasi sistem informasi P3M terintegrasi [7], hingga arsitektur modern seperti *microservices* [1] dan aplikasi permainan menggunakan metrik *maintainability* yang dinamis [9]. Bahkan, studi eksperimental terbaru di ranah global telah memperluas cakupan ini dengan menguji kapabilitas otomatisasi *refactoring* berbasis kecerdasan artifisial untuk mereduksi *code smells* secara sistematis pada puluhan proyek sumber terbuka (*open-source*) [14].

Namun, terdapat celah penelitian di mana sebagian besar studi *refactoring* dan evaluasi kompleksitas sebelumnya masih berfokus pada sistem informasi berskala menengah, aplikasi permainan [9], studi kasus modularitas generik [13], platform lokal [15], atau evaluasi otomatisasi berskala makro [14]. Belum ditemukan kajian empiris yang secara spesifik mengukur dampak teknis dari kombinasi teknik *Extract Method* dan *Pull Up Method* secara manual pada hirarki kelas event di sistem LMS masif berbasis PHP seperti Moodle versi 5.1.2. Meskipun manfaat umum peningkatan efektivitas kode melalui *refactoring* telah banyak didokumentasikan [13], evaluasi kuantitatif menggunakan standar metrik terbaru untuk mengoptimalkan distribusi tanggung jawab kelas pada program PHP berskala besar masih sangat terbatas [2].

Kebaruan dari penelitian ini terletak pada analisis komparatif kuantitatif yang difokuskan pada penataan ulang struktur hierarki kelas *event* di Moodle versi 5.1.2 dengan mengacu pada prinsip-prinsip pemeliharaan formal dalam Software Engineering Body of Knowledge (SWEBOK) [12]. Penelitian ini tidak hanya mengukur dampak *refactoring* secara luas, tetapi melakukan bedah mendalam pada lima kelas *event* utama, yaitu base, assessable_submitted, submission_viewed, feedback_viewed, dan grading_form_viewed. Melalui pengukuran metrik *Weighted Methods per Class* (WMC) dan *Lines of Code* (LOC) pada tingkat kelas, serta *Cyclomatic Complexity Number* (CCN) pada tingkat metode menggunakan alat analisis statis PDepend, penelitian ini memberikan kontribusi orisinal dalam membuktikan bagaimana teknik *Pull Up Method* dan *Extract Method* secara manual dapat mendistribusikan kembali kompleksitas kode serta memangkas jalur keputusan logis pada sistem *enterprise* yang telah sangat matang.

2. Metode Penelitian

2.1 Pemeliharaan Perangkat Lunak

Pemeliharaan perangkat lunak merupakan fase yang sangat penting dalam siklus hidup pengembangan sistem guna menjaga fungsionalitas dan relevansi program terhadap kebutuhan pengguna yang dinamis. Seiring dengan berjalannya waktu, kode sumber cenderung mengalami penurunan kualitas akibat penambahan fitur yang tidak terstruktur, yang pada akhirnya meningkatkan beban kerja pengembang dalam melakukan perbaikan atau modifikasi.

2.1 Objek Penelitian

Objek penelitian ini adalah modul assignment (mod_assign) pada sistem Moodle versi 5.1.2. Fokus analisis diarahkan pada lima kelas spesifik yang menangani logika *event* dan distribusi data submisi, yaitu: base.php, assessable_submitted.php,

submission_viewed.php, feedback_viewed.php, dan grading_form_viewed.php. Pemilihan kelas-kelas ini didasarkan pada perannya sebagai unit analisis utama yang mewakili struktur hierarki dan interaksi antar objek dalam modul tersebut.

2.2 Alat dan Instrumen Penelitian
Analisis kode dilakukan menggunakan metode analisis statis, di mana kode sumber diperiksa tanpa melakukan eksekusi program. Alat utama yang digunakan adalah PDepend, sebuah instrumen analisis statis yang mampu mengekstraksi berbagai metrik kompleksitas dan kualitas kode berbasis PHP secara otomatis.

2.3 Metrik Pengukuran Kompleksitas
2.3.1 Weighted Methods per Class (WMC)
Metrik WMC digunakan untuk mengukur total kompleksitas dari seluruh metode yang didefinisikan dalam suatu kelas. WMC dihitung dengan menjumlahkan kompleksitas masing-masing metode yang ada di dalam kelas tersebut. Jika setiap metode diberi bobot kompleksitas sebesar c_i, maka nilai WMC untuk suatu kelas dapat dirumuskan sebagai berikut:

$$WMC = \sum_{i=1}^{n} c_i$$

Di mana:
n: Jumlah total metode dalam kelas tersebut.
c_i: Nilai kompleksitas (biasanya nilai CCN) dari metode ke-i.

Nilai WMC yang tinggi menunjukkan bahwa kelas tersebut memiliki tanggung jawab fungsional yang besar, sehingga lebih sulit untuk dipelihara dan berisiko tinggi terhadap munculnya kesalahan.

2.3.2 Cyclomatic Complexity Number (CCN)
Cyclomatic Complexity adalah metrik yang digunakan untuk mengukur kompleksitas logis dari suatu program dengan menghitung jumlah jalur linier yang independen melalui kode sumber. Metrik ini didasarkan pada representasi grafik alur kontrol (*control flow graph*) program. Rumus umum untuk menghitung CCN adalah:

$$M = E - N + 2P$$

Atau dalam bentuk yang lebih sederhana untuk struktur kode sekuensial:

$$V(G) = P + 1$$

Di mana:
M atau V(G): Nilai kompleksitas siklomatik.
E: Jumlah *edges* (jalur antar titik) dalam grafik alur.
N: Jumlah *nodes* (titik keputusan/pernyataan) dalam grafik alur.
P: Jumlah titik keputusan (*predicate nodes*) seperti pernyataan if, while, for, dan case.

Dalam konteks penelitian ini, penurunan nilai CCN setelah refactoring menandakan bahwa alur logika kode telah disederhanakan, yang mempermudah proses pengujian unit (*unit testing*).

2.3.3 Logical Lines of Code (LOC)
LOC didefinisikan sebagai jumlah total baris dalam file kode sumber, yang mencakup pernyataan eksekusi, namun dalam penelitian ini difokuskan pada baris yang memberikan kontribusi pada struktur.

2.4 Teknik Refactoring yang Diterapkan
Dua teknik refactoring utama diterapkan dalam penelitian ini untuk membedah kompleksitas kode:
1. Extract Method: Digunakan untuk memecah metode yang terlalu panjang menjadi metode-metode baru yang lebih kecil dan fokus pada satu tanggung jawab spesifik. Dalam penelitian ini, teknik ini diterapkan pada metode *factory* create_from_... untuk mengisolasi proses persiapan data ke dalam metode prepare_event_data().
2. Pull Up Method: Digunakan untuk memindahkan metode atau logika yang bersifat redundan dari kelas anak ke kelas induk (*base class*). Teknik ini bertujuan untuk meningkatkan penggunaan kembali kode (*reusability*) dan menyederhanakan struktur kelas-kelas turunan.

2.5 Tahapan Eksperimen
Eksperimen dilakukan melalui empat tahapan utama:
1. Pengukuran awal, melakukan ekstraksi metrik awal pada kelima kelas target menggunakan PDepend untuk mendapatkan data *Weighted Methods per Class* (WMC), *Lines of Code* (LOC), dan *Cyclomatic Complexity Number* (CCN).

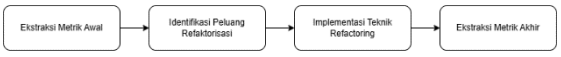
2. Identifikasi peluang *refactoring*, pada tahap identifikasi, fokus utama penelitian adalah mendeteksi keberadaan *code smells*, yaitu pola struktur kode program yang buruk sehingga terindikasi adanya suatu masalah dan membuat programmer sulit untuk memahami, membaca dan memodifikasi kode ketika akan melakukan perubahan, khususnya *God Class*. *God Class* merujuk pada kelas yang terlalu dominan, memiliki ribuan baris kode, dan menangani terlalu banyak tanggung jawab fungsional. Dalam modul assignment Moodle, kelas utama seperti assign bertindak sebagai *God Class* yang sangat berisiko untuk dimodifikasi secara langsung dalam skala kecil.

   Untuk menghindari kompleksitas yang tidak terkendali, penelitian ini secara sengaja mengalihkan unit analisis dari *God Class* ke

kelas-kelas event yang lebih modular namun memiliki ketergantungan logika yang tinggi, seperti base.php dan kelas turunan lainnya. Identifikasi dilakukan dengan mencari metode dengan duplikasi logika validasi yang identik di beberapa file berbeda.

3. Implementasi, menerapkan teknik *Extract Method* dan *Pull Up Method*.

4. Pengukuran akhir, melakukan pengukuran ulang menggunakan alat dan parameter yang sama untuk mendapatkan data pasca-refactoring.



Gambar 1. Tahapan Eksperimen

2.6 Metrik Pengukuran

Keberhasilan refactoring dinilai berdasarkan tiga metrik utama:

1. Lines of Code (LOC): Mengukur jumlah baris kode untuk melihat *overhead* struktural yang dihasilkan.
2. Weighted Methods per Class (WMC): Mengukur total kompleksitas pada level kelas untuk menilai beban logika secara keseluruhan.
3. Cyclomatic Complexity Number (CCN): Mengukur jumlah jalur linier yang independen dalam alur kontrol kode untuk menentukan tingkat kemudahan pengujian dan keterpahaman kode.

3. Hasil dan Pembahasan

3.1 Implementasi Teknik Pull Up Method

Teknik ini diterapkan untuk menghilangkan redundansi logika validasi assignid yang ditemukan di seluruh kelas *event* turunannya. Logika tersebut ditarik ke dalam kelas induk (base.php) untuk mencapai sentralisasi kontrol.

base.php sebelum *refactoring* tidak terdapat kode logika validasi assignid, setelah refactoring:

base.php

```
    protected function validate_assignid() {
        if (!isset($this->other['assignid'])) {
            throw new \coding_exception('The \'assignid\' value must be set in other.');
        }
    }
```

Potongan kode validasi assignid pada feedback_viewed.php sebelum *refactoring*:

feedback_viewed.php

```
    if (!isset($this->other['assignid'])) {
        throw new \coding_exception('The \'assignid\' value must be set in other.');
    }
```

feedback_viewed.php setelah *refactoring*:

feedback_viewed.php

```
$this->validate_assignid();
```

penggunaan $this->validate_assignid() pada kelas anak menggantikan blok pengkondisian manual. Hal ini menjelaskan mengapa nilai CCN pada feedback_viewed dan grading_form_viewed menurun (masing-masing 3 menjadi 2 dan 4 menjadi 3), karena jalur keputusan kini diarahkan ke kelas induk. Sebaliknya, kelas base mengalami kenaikan CCN (7 menjadi 8) karena kini memikul beban validasi secara kolektif untuk seluruh modul.

3.2 Implementasi Teknik Extract Method

Teknik *Extract Method* diterapkan pada metode *factory* create_from_submission() untuk mematuhi prinsip *Single Responsibility*. Berikut adalah metode create_from_submission() pada kelas submission_viewed.php sebelum *refactoring*:

submission_viewed.php

```
    public static function create_from_submission(\assign $assign, \stdClass $submission) {
        $data = array(
            'objectid' => $submission->id,
            'relateduserid' => $submission->userid,
            'context' => $assign->get_context(),
            'other' => array(
                'assignid' => $assign->get_instance()->id,
            ),
        );
        /** @var submission_viewed $event */
        $event = self::create($data);
        $event->set_assign($assign);

        $event->add_record_snapshot('assign_submission', $submission);
        return $event;
    }
```

Metode create_from_submission() pada submission_viewed.php setelah *refactoring*:

submission_viewed.php

```
    public static function create_from_submission(\assign $assign, \stdClass $submission) {
        $data = self::prepare_event_data($assign, $submission);
        /** @var submission_viewed $event */
        $event = self::create($data);
        $event->set_assign($assign);
        $event->add_record_snapshot('assign_submission', $submission);
        return $event;
    }
    protected static function prepare_event_data(\assign $assign, \stdClass $submission) {
        $data = array(
            'objectid' => $submission->id,
            'relateduserid' => $submission->userid,
            'context' => $assign->get_context(),
            'other' => array(
                'assignid' => $assign->get_instance()->id,
            ),
        );
```

```
    return $data;
}
```

Implementasi di atas memberikan penjelasan teknis terhadap kenaikan metrik LOC sebesar 10 baris pada submission_viewed dan 13 baris pada assessable_submitted. Penambahan baris kode ini merupakan *overhead* struktural yang diperlukan untuk mendefinisikan deklarasi metode baru (protected static function...) dan merapikan struktur data. Namun, secara kualitas, hal ini menurunkan CCN pada metode utama (3 menjadi 2) karena alur logika persiapan data telah diisolasi ke dalam metode pembantu.

3.3 Hasil Pengukuran Metrik

Berdasarkan pengujian menggunakan PDepend, diperoleh data perbandingan metrik untuk lima kelas target sebagaimana disajikan pada Tabel 1.

Tabel 1. Distribusi Metrik Kualitas pada Tingkat Kelas dan Metode.

Nama Kelas	Nama Method	Metrik	Sebelum	Sesudah
base	(Level Kelas)	WMC	10	12
	(Level Kelas)	LOC	84	95
	set_assign	CCN	4	4
	get_assign	CCN	3	3
	get_url	CCN	1	1
	validate_data	CCN	2	2
	validate_assignid	CCN	0 (method tidak ada)	2
feedback_viewed	(Level Kelas)	WMC	9	8
	(Level Kelas)	LOC	86	84
	create_from_grade	CCN	1	1
	init	CCN	1	1
	get_name	CCN	1	1
	get_description	CCN	1	1
	validate_data	CCN	3	2
	get_objectid_mapping	CCN	1	1
	get_other_mapping	CCN	1	1
submission_viewed	(Level Kelas)	WMC	8	9
	(Level Kelas)	LOC	86	96
	prepare_event_data	CCN	0 (method tidak ada)	1
	create_from_submission	CCN	1	1
	init	CCN	1	1
	get_name	CCN	1	1
	get_description	CCN	1	1
	validate_data	CCN	2	2
	get_objectid_mapping	CCN	1	1
	get_other_mapping	CCN	1	1
assessable_submitted	(Level Kelas)	WMC	10	11
	(Level Kelas)	LOC	92	105
	prepare_event_data	CCN	0 (method tidak ada)	3
	create_from_submission	CCN	3	1
	get_description	CCN	1	1
	get_name	CCN	1	1
	init	CCN	1	1
	validate_data	CCN	2	2
	get_objectid_mapping	CCN	1	1
	get_other_mapping	CCN	1	1

grading_form_viewed	(Level Kelas)	WMC	9	8
	(Level Kelas)	LOC	91	89
	create_from_user	CCN	1	1
	init	CCN	1	1
	get_name	CCN	1	1
	get_description	CCN	1	1
	validate_data	CCN	3	2
	get_other_mapping	CCN	1	1

3.4 Analisis Efisiensi pada Kelas Turunan

Hasil penelitian menunjukkan bahwa teknik *Pull Up Method* berhasil memberikan efisiensi ganda pada kelas feedback_viewed dan grading_form_viewed. Pada kedua kelas tersebut, terjadi penurunan pada seluruh metrik tingkat kelas (*LOC* berkurang 2 baris, dan *WMC* berkurang 1 poin). Secara mikro, efisiensi ini dipicu oleh penurunan kompleksitas pada metode validate_data() yang semula bernilai *CCN* = 3 menjadi *CCN* = 2. Hal ini membuktikan bahwa pemindahan logika pemeriksaan assignid ke kelas induk mengurangi beban kerja internal kelas anak, mengeliminasi redundansi kode, serta membuat struktur kelas anak menjadi lebih ramping dan mudah dipahami.

3.5 Analisis Trade-off pada Extract Method

Pada kelas submission_viewed dan assessable_submitted, terlihat adanya *trade-off* yang jelas antara volume fisik kode dengan penyederhanaan alur logika metode utama. Penggunaan teknik *Extract Method* menyebabkan kenaikan *LOC* (hingga +13 baris pada assessable_submitted dan +10 baris pada submission_viewed) serta kenaikan *WMC* sebanyak +1 poin akibat lahirnya metode baru bernama prepare_event_data(). Namun, pada kelas assessable_submitted, nilai *CCN* pada metode utama create_from_submission() berhasil ditekan dari 3 menjadi 1. Jalur keputusan yang rumit diisolasi sepenuhnya ke dalam metode pembantu. Hal ini menunjukkan bahwa peningkatan jumlah baris kode deklaratif (seperti *method signature*, tanda kurung kurawal, dan *return statement*) merupakan biaya struktural (*overhead*) yang sepadan untuk mendapatkan metode utama yang jauh lebih bersih, linear, dan terisolasi dari risiko *bug*.

3.6 Peran Kelas Induk dalam Sentralisasi Kompleksitas

Kelas base mengalami peningkatan nilai pada metrik tingkat kelas, yaitu *WMC* (+2) dan *LOC* (+11). Fenomena ini mengonfirmasi keberhasilan penerapan prinsip Inheritance dan orientasi objek yang baik. Peningkatan ini dipicu langsung oleh penambahan metode baru validate_assignid() yang memiliki nilai *CCN* = 2. Sebagai kelas induk, base secara sengaja dikonfigurasi untuk menyerap kompleksitas validasi yang sebelumnya tersebar secara redundan di berbagai kelas anak demi memastikan konsistensi aturan bisnis di seluruh modul *assignment*. Meskipun beban pada kelas base bertambah, arsitektur ini jauh lebih menguntungkan bagi pemeliharaan sistem jangka panjang karena jika ada perubahan aturan validasi di masa depan, pengembang hanya perlu melakukan modifikasi pada satu titik sentral ini saja.

4. Kesimpulan

Penelitian ini membuktikan bahwa penerapan teknik *Extract Method* dan *Pull Up Method* pada modul *assignment* Moodle versi 5.1.2 memberikan dampak pada dimensi makro (tingkat kelas) dan mikro (tingkat metode). Berdasarkan fakta eksperimen, implementasi *Pull Up Method* mereduksi beban kerja kelas anak, di mana kelas feedback_viewed dan grading_form_viewed mengalami penurunan metrik *Weighted Methods per Class* (WMC) dan *Lines of Code* (LOC) secara simultan, yang dipicu langsung oleh penurunan kompleksitas metode validate_data() dari nilai *Cyclomatic Complexity Number* (CCN) 3 menjadi 2. Di sisi lain, penggunaan *Extract Method* pada kelas submission_viewed dan assessable_submitted memberikan trade-off berupa peningkatan volume fisik kode (LOC hingga +13 baris) dan kenaikan nilai WMC kelas akibat overhead struktural dari deklarasi metode baru prepare_event_data(). Namun, secara mikro teknik ini sukses menekan kompleksitas dengan memangkas nilai CCN metode utama create_from_submission() dari 3 menjadi 1 pada kelas assessable_submitted, serta mempertahankan stabilitas alur linear pada kelas submission_viewed. Kenaikan metrik pada kelas induk (base) melalui kehadiran metode baru validate_assignid() dengan nilai CCN 2 mengonfirmasi adanya sentralisasi kompleksitas dari anak ke induk sebagai konsekuensi logis untuk mencapai konsistensi validasi di tingkat sistem. Secara keseluruhan, *refactoring* berimplikasi pada peningkatan keterpahaman kode dan kemudahan pengujian melalui distribusi logika yang lebih modular. Temuan ini dapat diaplikasikan oleh praktisi pengembang perangkat lunak sebagai landasan strategi dalam mengelola sistem skala besar, di mana peningkatan jumlah baris kode dapat diterima sebagai kompensasi atas alur logika metode yang lebih sederhana dan terorganisir.

Berdasarkan hasil yang diperoleh, penelitian selanjutnya disarankan untuk mengeksplorasi metrik keterkaitan (*coupling*) dan kelekatan (*cohesion*) guna

memahami dampak *refactoring* terhadap hubungan antar-objek secara lebih mendalam setelah dilakukannya sentralisasi logika pada kelas induk. Selain itu, penting untuk dilakukan pengujian regresi fungsional yang lebih luas untuk memastikan bahwa dekomposisi dan penyederhanaan alur metode tidak memengaruhi kinerja atau waktu respons sistem saat menangani volume data submisi yang besar. Penelitian di masa depan juga dapat memperluas objek kajian pada modul-modul Moodle yang lebih baru atau modul inti lainnya untuk melihat konsistensi dampak refactoring pada arsitektur perangkat lunak yang menggunakan pola desain (*design pattern*) yang lebih modern..

Daftar Rujukan

[1] Hakim, A. F. N., & Prismana, I. G. L. P. E. (2023). Refactoring Arsitektur Microservice Pada Aplikasi Absensi PT. Graha Usaha Teknik. Jurnal Informatika Dan Sistem Informasi (JIFOSI), 4(1), 38-45.

[2] Mardiana, B. D., & Rochimah, S. (2024). Large Class Refactoring in PHP Program to Improve Modularity Based on ISO/IEC 25023 Metrics. 2024 IEEE 10th Information Technology International Seminar (ITIS), 1-6. https://doi.org/10.1109/ITIS63013.2024.10845243.

[3] Aldya, A. P., & Hartono, H. L. G. S. (2022). Peningkatan Kualitas Maintainability Sistem PPID DINKES Cimahi Menggunakan Metode Refactoring. Jurnal Tekno Kompak, 16(1), 121-133. https://doi.org/10.33365/jtk.v16i1.1548.

[4] Panjaitan, L. A. S., Aldya, A. P., & Hartono, H. L. G. S. (2022). Optimalisasi Kualitas Integrity Pada Sistem Dinas Kesehatan Kota Cimahi Menggunakan Teknik Refactoring. Jurnal Ilmiah Teknologi Informasi Terapan (JITTER), 8(2), 241-247. https://doi.org/10.33197/jitter.vol8.iss2.2022.810.

[5] Setiawan, H., & Emanuel, A. W. R. (2021). Pengaruh Refactoring Extract Method terhadap Pengembangan Aplikasi menggunakan Test Driven Development. Jurnal RESTI (Rekayasa Sistem Dan Teknologi Informasi), 5(5), 903 - 910. https://doi.org/10.29207/resti.v5i5.3400.

[6] Sitinjak, D. A. N., & Suwita, J. (2021). Metode Refactoring Untuk Meningkatkan Kualitas Maintainability Pada Sistem Informasi Puskesmas. Jurnal Informatika Terpadu (JIT), 7(1), 1 - 8. https://doi.org/10.54914/jit.v7i1.303.

[7] Sudrajat, S., & Kurniawan, D. (2021). Pengembangan Sistem Informasi P3M Terintegrasi Melalui Refactoring dan Penambahan Fitur dengan Metode R&D. Informatic: Jurnal Ilmu Komputer, 17(2), 108-118. https://doi.org/10.52958/iftk.v17i2.2355.

[8] Singh, R. S. K., & Kaswidjanti, W. (2020). Optimalisasi Perangkat Lunak Menggunakan Metode Refactoring. Jurnal Informatika, 14(1), 18-27. https://doi.org/10.31315/inf.v14i1.3468.

[9] Sadiyah, S. H., & Qoiriah, A. (2024). Pengaruh Refactoring Terhadap Tingkat Pemeliharaan Perangkat Lunak Pada Aplikasi Permainan "Infection Defender". JINACS: Journal of Informatics and Computer Science, 5(03), 400-413. https://doi.org/10.26740/jinacs.v5n03.p400-413.

[10] Ghifari, R. B. M., Fitri, S., Ardhyandoko, A., & Yaqin, M. A. (2024). Analisis dan Perancangan Software Pengukuran Matriks Skala dan Kompleksitas Kode Program. JACIS: Journal Automation Computer Information System, 4(1), 42–49. https://doi.org/10.47134/jacis.v4i1.72

[11] Khaleel, S. I., & Al-Khatouni, G. K. (2023). Measuring Maintainability Index Before and After Code Refactoring. MINAR International Journal of Applied Sciences and Technology, 5(3), 18-36.

[12] Washizaki, H. (Ed.). (2025). Guide to the Software Engineering Body of Knowledge (SWEBOK Guide), Version 4.0. IEEE Computer Society.

[13] Almogahed, A., Othman, M., Omar, M., Barraood, S. O., & Gilal, A. R. (2025). Improving Code Effectiveness Through Refactoring: A Case Study. Journal of Informatics and Web Engineering (JIWE), 4(3), 426–440. https://doi.org/10.33093/jiwe.2025.4.3.26.

[14] Cordeiro, J., Noei, S., & Zou, Y. (2024). An Empirical Study on the Code Refactoring Capability of Large Language Models. https://doi.org/10.48550/arXiv.2411.02320

[15] Lestari, W. F., Chrisnanto, Y. H., & Yuniarti, R. (2024). Teknik Refactoring untuk Kualitas Usability Sistem Informasi SPP dan Tunggakan SDIT Nuralima. Reslaj: Religion Education Social Laa Roiba Journal, 6(8), 4066–4078. https://doi.org/10.47476/reslaj.v6i8.3388.