Analisis Pengaruh *Design Pattern* Terhadap Pemeliharaan Perangkat Lunak *Learning Management System*

Albertus Kevin[#1], Hans Christian Kurniawan[#2]

[#]*Program Studi Informatika, Institut Teknologi Harapan Bangsa*
Jalan Dipatiukur No. 80-84, Bandung, Indonesia 40132
[1]vinalbertus@gmail.com
[2]hans_christian@ithb.ac.id

Abstract*— Software development focuses more on functionality so code quality is neglected. Design patterns can support faster development while supporting good code quality thus affecting long-term software maintenance. This research focuses on analyzing the effect of design patterns on software maintenance with open-source learning management system software in terms of design pattern characteristics. Five design patterns were applied and analyzed to assess the implementation method and design pattern characteristics. Testing is done using PHP Metrics. There are nine metrics to measure the complexity, size, cohesion, and dependability of each class. Overall, the design pattern had a good impact. Method templates and mediators have a good impact on cohesion, size, dependability, and complexity. Singletons increased the number of classes. Builders and strategies don't have much impact on size and complexity. Each design pattern generates a new class, so the complexity, dependencies, and size of the code are abstracted into that class.

***Keywords*— software maintainability, design pattern, PHP Laravel, learning management system, software metric, object-oriented programming, template method, mediator, builder, strategy, singleton, complexity, maintainability index, coupling, line of code**

***Abstrak*— Pengembangan perangkat lunak lebih berfokus pada fungsionalitas sehingga kualitas kode diabaikan. Pola desain dapat mendukung pengembangan yang lebih cepat sambil mendukung kualitas kode yang baik sehingga mempengaruhi pemeliharaan perangkat lunak jangka panjang. Penelitian ini berfokus pada analisis pengaruh pola desain terhadap pemeliharaan perangkat lunak dengan perangkat lunak *open source learning management system* ditinjau dari karakteristik pola desain. Lima pola desain diterapkan kemudian dilakukan analisis untuk menilai metode implementasi dan karakteristik pola desain. Pengujian dilakukan dengan menggunakan *PHP Metrics*. Terdapat sembilan metrik untuk mengukur kompleksitas, ukuran, kohesi, dan dependabilitas setiap *class*. Secara keseluruhan, pola desain memiliki dampak yang baik. *Template method* dan mediator memiliki dampak yang baik pada kohesi, ukuran, ketergantungan, dan kompleksitas. *Singleton* menambah jumlah *class*. Pembangun dan strategi tidak memiliki banyak pengaruh dalam hal ukuran dan kompleksitas. Setiap pola desain menghasilkan *class* baru sehingga kompleksitas, ketergantungan, dan ukuran kode diabstraksikan ke dalam *class* tersebut.**

Kata Kunci*— pemeliharaan perangkat lunak, pola desain, PHP Laravel, sistem manajemen pembelajaran, metrik perangkat lunak, pemrograman berorientasi objek, metode *template*, mediator, *builder*, strategi, *singleton*, kompleksitas, indeks pemeliharaan, penggabungan, *coupling*, *line of code

I. PENDAHULUAN

Pengembangan perangkat lunak lebih berfokus pada fungsionalitas dan mengabaikan aspek jangka panjang seperti pemeliharaan. Padahal, aspek *maintainability* penting dalam proyek perangkat lunak berkepanjangan dengan estimasi biaya mencapai 60% dari keseluruhan proses perangkat lunak [1]. Berdasarkan hal tersebut, perlu ditentukan bagaimana arsitektur atau struktur kode selama proses pengembangan aplikasi. Selama ini banyak perusahaan memiliki struktur kode tertentu untuk mempermudah pengembangan aplikasi serta sebagai solusi dari setiap masalah kode selama proses pengembangan. Struktur kode tersebut disebut *design pattern* [1]–[3], meskipun belum didokumentasikan secara resmi. Hingga muncul terminologi *design pattern* sebagai solusi umum untuk mengatasi masalah struktur kode. *Design pattern* merupakan *best practice* yang berasal dari berbagai sumber, seperti pengalaman pengembang atau hasil penelitian [4].

Penerapan *design pattern* memiliki banyak manfaat, seperti peningkatan pemahaman program (*understandability*), perubahan yang dilakukan secara fleksibel (*flexibility*), penggunaan berulang komponen/kode (*reusability*), dan perluasan cakupan aplikasi (*extensibility*), serta mengurangi waktu untuk pengembangan perangkat lunak [5]. Namun, *design pattern* tidak selalu memberikan dampak positif, seperti menurunkan performa sistem dan meningkatnya kompleksitas seiring perkembangan aplikasi [6]. Pada penelitian [5] dinyatakan bahwa penerapan *GoF design pattern* berdampak terhadap tingkat kompleksitas program (*cyclomatic complexity*), namun meningkatkan ukuran kode seperti jumlah *class* dan *line of code* (LOC). Pada penelitian [1] penerapan *facade* dan *observer* memberikan dampak nilai *maintainability index* yang sangat baik, meskipun nilai *cyclomatic complexity* dan tingkat *coupling* hanya mengalami sedikit sekali perubahan. Sementara pada penelitian [2], menyebutkan bahwa penerapan *observer* memberikan tingkat kompleksitas yang cukup tinggi di beberapa kasus.

Berdasarkan hal tersebut, banyak penelitian yang mencoba menganalisis dampak *design pattern* terhadap atribut kualitas perangkat lunak. Beberapa penelitian tersebut memberikan yang berbeda-beda sehingga tidak mencapai kesimpulan yang pasti. Nanthaamornphong, Aziz, dkk. [4] dalam penelitiannya menyimpulkan bahwa salah satu faktor adalah banyak penelitian yang kurang konsisten mendefinisikan pengertian *software maintainability* dan tidak mengacu pada standar internasional yang ada [1], [4], [6], [7].

Pada penelitian ini dilakukan analisis dampak *design pattern* terhadap perangkat lunak dari aspek pemeliharaan perangkat lunak menggunakan acuan definisi standar internasional ISO dalam hal *software maintainability*. Berbagai *software metric* digunakan sebagai nilai acuan agar lebih komprehensif dan mewakili karakteristik pemrogramanan berorientasi objek serta memiliki asosiasi kuat dengan karakteristik *software maintainability*.

Selain itu, penelitian ini menggunakan objek *software* dengan kategori LMS karena proses bisnis dan pengembangan fitur yang bersifat dinamis. Atas dasar tersebut, pemilihan dan penerapan *design pattern* lebih mudah dilakukan serta membutuhkan kualitas kode yang baik agar mempermudah pengembangan fitur.

II. METODOLOGI

A. Object Oriented Programming

Pengembangan perangkat lunak yang memandang setiap komponen seperti sebuah objek. Terdapat 4 pilar utama dalam OOP [8], yaitu:

1. *Abstraction*: menyembunyikan kompleksitas program dengan merepresentasikan data yang dibutuhkan saja pada sebuah model. Misalnya, pesawat memiliki banyak atribut, seperti kapasitas mesin, bobot, dan banyaknya tempat duduk. Sebagai penumpang, tidak perlu mengetahui kapasitas mesin dan bobot sehingga model pesawat hanya merepresentasikan data-data yang dibutuhkan oleh penumpang saja.
2. *Encapsulation*: menyembunyikan bagian-bagian dari sebuah model dan menyediakan antarmuka yang dapat diakses oleh bagian program lainnya. Contohnya, model mobil memiliki fungsi menyalakan mesin. Pengguna, atau *client,* hanya perlu memanggil fungsi tersebut tanpa tahu apa yang dilakukan fungsi tersebut untuk menyalakan mesin.
3. *Polymorphism*: kemampuan sebuah objek memiliki banyak peran pada konteks tertentu selama dalam satu pohon pewarisan yang sama. Misalnya, seorang manajer di divisi A dikategorikan sebagai karyawan. Namun, ketika dia bertugas di divisi B, dia menjadi seorang direktur. Masih karyawan yang sama, namun memiliki peran yang berbeda di divisi yang berbeda.
4. *Inheritance*: pewarisan dengan membuat sebuah model sebagai anak dari model lainnya yang sudah ada sebelumnya. Terdiri dari *parent class* dan *child class*. *Child class* akan mewarisi semua data milik *parent class*. *Inheritance* dapat mendukung *polymorphism*.

B. Design Pattern

Design pattern yang sering digunakan dalam industri perangkat lunak adalah GoF (*gang of four*) yang dibagi menjadi 3 kategori [6], [8], [9]:

1. *Creational* ketika membuat objek baru. Contoh *design pattern* yang terkait adalah *builder*, *singleton*, *factory method*, *abstract factory*, dan *prototype*.
2. *Structural* merealisasikan hubungan antar *class*. *Design pattern* yang terkait adalah *bridge*, *adapter*, *composite*, *decorator*, *facade*, *flyweight*, dan *proxy*.
3. *Behavioural* mengatur cara komunikasi antar objek. Contohnya *mediator*, *observer*, *strategy*, *command*, *iterator*, *chain of responsibility*, *memento*, *template method*, *visitor*, dan *state*.

Dalam penelitian ini, *design pattern* yang diuji adalah *strategy*, *singleton*, *builder*, *template method*, dan *mediator*.

1) Strategy

Bertujuan untuk mendefinisikan beberapa algoritme yang serupa, yaitu memiliki tujuan yang sama dan spesifik, lalu ditempatkan ke dalam *class* yang terpisah sebagai objek tersendiri. Misalnya, algoritme membuat rute perjalanan yang berbeda untuk mobil, motor, atau angkutan umum. Tujuannya sama, yaitu membuat rute perjalanan, tetapi implementasinya berbeda untuk setiap jenis kendaraan.

2) Singleton

Bertujuan untuk memastikan sebuah *class* hanya memiliki satu objek dan menyediakan akses umum ke objek tersebut. Biasanya digunakan untuk mengendalikan akses ke sebuah sumber daya yang cukup berat, seperti koneksi basis data atau *file* yang dapat digunakan bersamaan.

3) Builder

Memungkinkan pembuatan objek kompleks per langkah sehingga dapat membuat beberapa objek dengan tipe yang berbeda menggunakan kode konstruksi yang sama.

4) Template Method,

Bertujuan untuk mendefinisikan kerangka algoritme sebagai *template* pada *superclass* dan memungkinkan beberapa langkah tertentu dapat diubah oleh *subclass* tanpa mengubah strukturnya.

5) Mediator

Bertujuan mengurangi ketergantungan antar objek dengan cara membatasi komunikasi yang dilakukan antar objek untuk mengharuskannya melewati objek penengah yang disebut mediator sehingga antar objek tetap dapat berkolaborasi, namun secara tidak langsung.

C. Software Metric

Software metric digunakan untuk melakukan penilaian terhadap kualitas perangkat lunak. Terdapat sembilan metrik yang digunakan.

1. *Weighted Method per Class* (WMC)
   Semakin besar jumlah *method* dalam sebuah *class*, semakin kecil kemungkinan *class* tersebut memenuhi

karakteristik reusability [7], [10]. Berikut ini adalah rumus WMC.

$$WMC = \sum_{i=1}^{n} C_i W_i \quad (1)$$

C_i adalah *class* ke-*i*, W_i adalah nilai bobot kompleksitas, dan *n* adalah jumlah *method*. Bobot dengan nilai 1 artinya hanya menghitung jumlah fungsi tanpa mempertimbangkan tingkat kompleksitas fungsi.

2. *Coupling*
   Semakin besar ketergantungan sebuah *class* dengan *class* lain akan menyulitkan jika dilakukan perubahan. Terdapat 3 metrik yang digunakan, *afferent coupling* (*Ca*), *efferent coupling* (*Ce*), dan *instability* (*I*). Nilai *I* merupakan hasil perhitungan rasio antara nilai *Ca* dan *Ce* [7], [10].

$$I = \frac{Ce}{Ca + Ce} \quad (2)$$

3. *Lack of Cohesion Method* (LCOM)
   Jumlah *disjoint* (tidak berpotongan) kumpulan fungsi dalam sebuah *class* bertujuan menghitung tingkat relasi antar fungsi dalam suatu *class*. Kompleksitas meningkat seiring dengan meningkatnya LCOM [10].
4. *Cyclomatic Complexity* (*CC* atau *G*).
   Menghitung kompleksitas program berdasarkan kemungkinan alur program yang bisa dilewati. Pada persamaan berikut, notasi *E* merupakan jalur program yang bisa dieksekusi, seperti percabangan atau perulangan. Notasi *N* merupakan banyaknya baris program yang akan dieksekusi. Notasi *P* merupakan jumlah titik akhir program kemungkinan akan berakhir, seperti *return* [2].

$$CC = E - N + 2p \quad (3)$$

5. *Lines of Code* (LOC).
   Jumlah baris kode yang bukan berupa komentar atau baris kosong [2].
6. *Halstead Volume* (*Hals Volume* atau *V*).
   Menghitung keseluruhan panjang program berdasarkan jumlah operator dan operan pada *source code*. Untuk dapat menghitung nilai *V*, diperlukan nilai *N* dan *n* [11].

$$\begin{aligned} N &= N_1 + N_2 \\ n &= n_1 + n_2 \\ V &= N .^2 \log n \end{aligned} \quad (4)$$

   N adalah jumlah total operan dan operator yang didapat dari penjumlahan total operator (N_1) dan total operan (N_2). Sementara *n* adalah jumlah total operator unik (n_1) dan operan yang unik (n_2).
7. *Maintainability Index* (MI).
   Model regresi yang melibatkan *cyclomatic complexity* (CC), LOC, dan *Halstead Volume* untuk menilai tingkat *maintainability* perangkat lunak. Beberapa komponen tambahan, seperti *perCM* merupakan persenan rata-rata jumlah baris komentar per *class* [12].

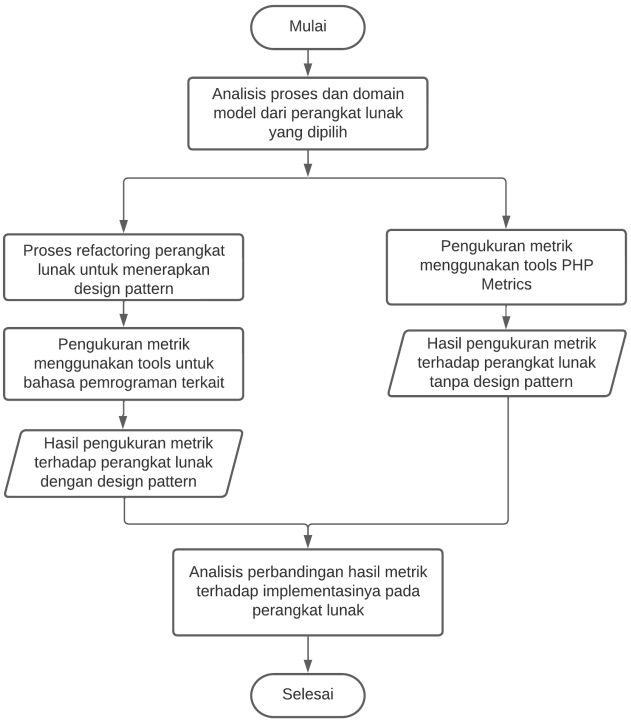
$$\begin{aligned} MI_{woc} &= 171 - 5{,}2\ln(V) - 0{,}23(G) - 16{,}2\ln(LOC) \\ MI_{cw} &= 50\sin\left(\sqrt{2{,}4 \times perCM}\right) \\ MI &= MI_{woc} + MI_{cw} \end{aligned} \quad (5)$$

D. Perancangan Sistem

1) Flowchart Global

Diawali menganalisis objek penelitian berupa aplikasi *web open source* untuk memahami proses bisnis, arsitektur, dan alur proses dari sistem. Langkah berikutnya menganalisis lokasi potensial penerapan *design pattern* disesuaikan dengan sistem. Pemilihan dilakukan secara manual. Setelah itu, dilakukan *refactoring*. *Source code* disimpan di *Github* agar dapat mengelola versi kode sebelum dan sesudah dilakukan penerapan *design pattern*. Setelah itu, dilakukan pengukuran menggunakan *tools PHP Metric* kemudian analisis perbandingan metrik. Terakhir menyimpulkan hasil sementara.

Selanjutnya, dilakukan analisis manual hasil metrik terhadap implementasi *design pattern* pada objek sehingga penelitian ini dapat menjelaskan secara menyeluruh sebab akibat atau mengapa nilai-nilai metrik mengalami perubahan. Rangkuman alur proses diperlihatkan pada Gambar 1.



Gambar 1 Diagram alur proses penelitian

2) Dataset

Objek penelitian adalah berupa perangkat lunak yang bersifat *open source* dengan dokumentasi yang cukup lengkap dan dapat diunduh melalui *website* resmi [13]. Aplikasi berupa *learning management system (LMS)* dengan versi 2.x. Aplikasi *unifiedtransform* merupakan proyek baru (2018) sehingga banyak potensi penambahan fitur dan masih diperbarui serta dibangun dengan teknologi versi yang masih baru, seperti *framework Laravel 8.x* dan *Bootstrap 5.x*. Struktur kode juga tersusun dengan baik sehingga mudah dipahami.

III. Hasil dan Pembahasan

Pengujian diawali dengan melakukan analisis sistem secara manual untuk menentukan *design pattern* yang dapat diterapkan pada studi kasus ini. Setelah ditentukan masalah dan lokasi penerapan *design pattern,* dilakukan implementasi *design pattern* tersebut. Dengan bantuan perangkat lunak *version control*, perbandingan hasil penerapan antar *design pattern* menjadi lebih mudah dilakukan. Setelah itu, dilakukan pengujian nilai metrik, dengan membandingkan metrik sebelum dan sesudah penerapan *design pattern*.

A. Builder Pattern

Penerapan *builder pattern* bertujuan menghilangkan beberapa fungsi pengambilan data dari basisdata dengan menggunakan model dengan parameter data yang berbeda-beda pada *MarkRepository*. Perubahan yang terjadi adalah bertambahnya *class* baru sebagai *class Builder*. Perubahan besar terjadi pada *class Repository*.

Terdapat kenaikan *WMC* pada *class MarkRepository*, padahal 4 fungsi telah diganti menjadi 1 fungsi saja. Hal ini menunjukkan bahwa bobot kompleksitas menjadi acuan dalam perhitungan *WMC*. Meskipun jumlah fungsi berkurang, fungsi yang menggantikan 4 fungsi sebelumnya menjadi lebih kompleks karena mengakomodasi algoritme 4 fungsi sebelumnya, sejalan dengan naiknya *CC* yang cukup jauh.

Nilai *LCOM* berkurang yang artinya tugas dari *class MarkRepository* dan *class MarkController* lebih dipersempit sehingga mendekati prinsip dari *single responsibility*. Pengurangan ini karena 4 fungsi yang digantikan tidak saling berkaitan sehingga metrik *LCOM* menganggap fungsi-fungsi tersebut tidak memiliki kesatuan tugas pada *class* tersebut.

Nilai *halstead volume* dan *LOC* meningkat karena 4 fungsi lainnya diganti dengan 1 fungsi. Kompensasinya adalah diperlukan logika yang lebih untuk dapat mengakomodasi fungsionalitas dari 4 fungsi sebelumnya. Kenaikan nilai *MI* juga dipengaruhi penambahan baris komentar. Jika komentar tersebut dihilangkan, maka nilai *MI* pun ikut memburuk (lihat Tabel I), sejalan dengan kenaikan kompleksitas dan ukuran program.

Class MarkController mengalami kenaikan serupa dikarenakan baris tambahan membungkus *filter* atau data yang dibutuhkan untuk dikirimkan ke fungsi di *class Repository*. Hasil metrik untuk penerapan *builder pattern* dapat dilihat pada Tabel II dan Tabel III.

A. Strategi Pattern

Penerapan *strategy pattern* bertujuan untuk memecah *class UserRepository* yang menanggung banyak tanggung jawab. Semakin banyak jenis pengguna (guru, murid, staf, dll.) dapat menyebabkan *class UserRepository* membengkak akibat bertambahnya fungsi. Hasil perubahan nilai metrik untuk pe-

Tabel I
MI Sebelum dan Setelah *Builder Pattern* Tanpa Baris Komentar

Class	Metrik	
	MI (Tanpa Komentar)	
	Bef	***Aft***
MarkController	54,15	52,3
MarkRepository	42,88	38,91
MarkRepositoryBuilder	-	45,4

Tabel II
Hasil Metrik Sebelum *Builder Pattern*

Class	Metrik									
	Halstead Volume		**LOC**		**MI**		**WMC**		**CC**	
	Bef	***Aft***	***Bef***	***Aft***	***Bef***	***Aft***	***Bef***	***Aft***	***Bef***	***Aft***
MarkController	3299,18	3802,67	185	197	54,15	54,71	28	28	22	22
MarkRepository	761,1	832,33	46	59	42,88	62,61	11	18	5	15
MarkRepositoryBuilder	-	255,33	-	53	-	45,4	-	10	-	1

Tabel III
Hasil Metrik Setelah *Builder Pattern*

Class	Metrik							
	LCOM		***Ca***		***Ce***		***Instability***	
	Bef	***Aft***	***Bef***	***Aft***	***Bef***	***Aft***	***Bef***	***Aft***
MarkController	2	1	0	0	13	13	1	1
MarkRepository	7	4	1	1	5	6	0,83	0,86
MarkRepositoryBuilder	-	1	-	1	-	0	-	0

nerapan *strategy pattern* dapat dilihat pada Tabel IV dan Tabel V. Secara keseluruhan, dampaknya tidak begitu baik terhadap ukuran program, meskipun kenaikannya sedikit.

Perubahan nilai *LOC* dan *halstead volume* mempengaruhi nilai *MI*. Kenaikan ukuran kode dikarenakan setiap kali memanggil fungsionalitas untuk manipulasi data pengguna perlu mengatur konteks terlebih dahulu. Semakin sering berhubungan dengan fungsionalitas data pengguna, semakin banyak baris untuk mengatur konteks. Hal ini berarti menambah juga operan dan operator program yang mempengaruhi nilai *halstead volume*. Penurunan angka *Halstead volume* di beberapa *class* tertentu tidak diakibatkan dari *strategy pattern*, namun karena gaya penulisan kode. Sebelumnya, pada parameter *construct* terdapat tambahan parameter yang disebabkan penggunaan *provider* untuk menggunakan *UserRepository*. Namun, pada *strategy pattern*, *class UserRepository* dihilangkan lalu digantikan dengan *class strategy* sehingga operan dan operator berkurang. Dampak ini dapat diabaikan karena tidak berhubungan dengan *pattern*.

Nilai metrik *WMC*, *CC*, dan *LCOM* tidak mengalami perubahan. Untuk *class* model *User* terjadi peningkatan *Ca* yang artinya *class* ini digunakan oleh *class* lainnya dan penambahan *class*-nya adalah 2. Artinya, ada 2 *class* baru yang menggunakan model *User*, yaitu *class TeacherStrategy* dan *StudentStrategy*. Hal ini dikarenakan setiap jenis pengguna yang ada akan dibuatkan *class* baru sehingga pasti akan mengakses model *User* tersebut. Adapun 1 *class* lainnya adalah sebagai pengganti pemanggil model *User* yang sebelumnya berasal dari *UserRepository*, namun *class* ini dipecah setelah *strategy*. Berada di *class ContextUserRepository* untuk kode ganti *password* dikarenakan logika umum untuk semua jenis pengguna sehingga dapat diletakkan di *parent class* dari *class* konkret *strategy*. Perubahan nilai *Ca* mengakibatkan perubahan nilai *instability* yang menurun mendekati 0. Artinya *class* model menjadi stabil meskipun sebelumnya sudah berada di rentang yang baik (0 – 0,3).

TABEL IV
HASIL METRIK SEBELUM *STRATEGY PATTERN*

Class	**Metrik**									
	Halstead Volume		**LOC**		**MI**		**WMC**		**CC**	
	Bef	***Aft***	***Bef***	***Aft***	***Bef***	***Aft***	***Bef***	***Aft***	***Bef***	***Aft***
MarkController	3299,18	3319,42	185	187	54,15	53,89	28	28	22	22
UserController	1069,71	1144,21	112	123	57,56	55,45	17	17	6	6
AttendanceController	1049,59	1081,07	94	97	74,59	73,77	15	15	10	10
StudentStrategy	-	950,05	-	88	-	53,17	-	19	-	12
PromotionController	977,37	999,83	71	73	76,79	76,07	11	11	8	8
AcademicSettingController	505,23	495,16	55	56	74,64	74,29	6	6	3	3
TeacherStrategy	-	307,71	-	41	-	46,72	-	8	-	5
HomeController	212	237,07	27	31	94,12	90,57	2	2	1	1
ContextUserRepository	-	63	-	29	-	55,37	-	6	-	1
UpdatePasswordController	95,18	85,11	27	28	69,21	68,94	5	5	3	3
UserRepoStrategy	-	323,33	-	25	-	51,39	-	9	-	4
App\Models\User	144,75	144,75	20	20	105,96	105,96	3	3	1	1
App\Traits\StrategyContext	-	7,92	-	15	-	67,78	-	2	-	2

TABEL V
HASIL METRIK SETELAH *STRATEGY PATTERN*

Class	**Metrik**							
	LCOM		***Ca***		***Ce***		***Instability***	
	Bef	***Aft***	***Bef***	***Aft***	***Bef***	***Aft***	***Bef***	***Aft***
MarkController	2	2	0	0	13	13	1	1
UserController	1	1	0	0	10	10	1	1
AttendanceController	3	3	0	0	9	9	1	1
StudentStrategy	-	3	-	1	-	9	-	0,9
PromotionController	1	1	0	0	7	7	1	1
AcademicSettingController	1	1	0	0	10	10	1	1
TeacherStrategy	-	3	-	1	-	4	-	0,8
HomeController	1	1	0	0	6	6	1	1
ContextUserRepository	-	1	-	7	-	1	-	0,13
UpdatePasswordController	2	2	0	0	4	4	1	1
UserRepoStrategy	-	6	-	3	-	3	-	0,5
App\Models\User	2	2	4	6	1	1	0,2	0,14
App\Traits\StrategyContext	-	1	-	0	-	2	-	1

Berdasarkan nilai metrik, *strategy pattern* berdampak tidak begitu baik. Perubahan keseluruhan nilai metrik memburuk pada bagian ukuran kode dan jumlah *class* bertambah. Jika ada penambahan fitur pada aplikasi, *strategy pattern* dapat memberikan dampak yang baik. Kontrak *interface* yang sama dapat digunakan *client* sehingga cukup menambah *class* baru. Implementasi berbeda dan pemanggilan fungsi dengan nama yang sama menjadikan kode lebih representatif dan terprediksi. Misalnya, jika ingin menambah data staf perpustakaan atau staf administrasi, sintaks atau *interface* fungsi yang digunakan sudah pasti adalah *create*, sama ketika ingin menambahkandata guru dan murid. Namun, prediksi penambahan fitur seperti hal tersebut tidak dapat ditangkap oleh metrik-metrik yang bersifat statis.

B. Strategy Pattern vs. Singleton Pattern

Penerapan *singleton pattern* dapat dilakukan setelah penerapan *strategy pattern*. Hasil nilai metrik untuk penerapan *singleton pattern* dapat dilihat pada Tabel VI dan Tabel VII. Secara kode, tidak akan ada perubahan apapun, hanya menambah *class* baru. Secara tingkat kohesi, *coupling*, dan ukuran kode dari *class* terkait tidak mengalami perubahan. Hal ini dikarenakan *singleton* mendelegasikan pembuatan objek baru untuk disimpan di memori ke *class* khusus, yaitu *singleton* sehingga yang sebelumnya pembuatan objek merupakan tugas dari *traits StrategyContext*, dipindahkan ke *class* baru, *Singleton*.

TABEL VI
HASIL METRIK *STRATEGY* SEBELUM PENERAPAN *SINGLETON PATTERN*

Class	**Metrik**									
	Halstead Volume		**LOC**		**MI**		**WMC**		**CC**	
	Bef	***Aft***	***Bef***	***Aft***	***Bef***	***Aft***	***Bef***	***Aft***	***Bef***	***Aft***
MarkController	3319,42	3319,42	187	187	53,89	53,89	28	28	22	22
UserController	1144,21	1144,21	123	123	55,45	55,45	17	17	6	6
AttendanceController	1081,07	1081,07	97	97	73,77	73,77	15	15	10	10
StudentStrategy	950,05	950,05	88	88	53,17	53,17	19	19	12	12
PromotionController	999,83	999,83	73	73	76,07	76,07	11	11	8	8
AcademicSettingController	495,16	495,16	56	56	74,29	74,29	6	6	3	3
TeacherStrategy	307,71	307,71	41	41	46,72	46,72	8	8	5	5
HomeController	237,07	237,07	31	31	90,57	90,57	2	2	1	1
ContextUserRepository	63	63	29	29	55,37	55,37	6	6	1	1
UpdatePasswordController	85,11	85,11	28	28	68,94	68,94	5	5	3	3
UserRepoStrategy	323,33	323,33	25	25	51,39	51,39	9	9	4	4
App\Models\User	144,75	144,75	20	20	105,96	105,96	3	3	1	1
App\Traits\StrategyContext	7,92	7,92	15	15	67,78	67,78	2	2	2	2
SingletonStudentStrategyRepository	-	0	-	12	-	171	-	2	-	2
SingletonTeacherStrategyRepository	-	0	-	12	-	171	-	2	-	2

TABEL VII
HASIL METRIK *STRATEGY* SETELAH PENERAPAN *SINGLETON PATTERN*

Class	**Metrik**							
	LCOM		***Ca***		***Ce***		***Instability***	
	Bef	***Aft***	***Bef***	***Aft***	***Bef***	***Aft***	***Bef***	***Aft***
MarkController	2	2	0	0	13	13	1	1
UserController	1	1	0	0	10	10	1	1
AttendanceController	3	3	0	0	9	9	1	1
StudentStrategy	3	3	1	1	9	9	0,9	0,9
PromotionController	1	1	0	0	7	7	1	1
AcademicSettingController	1	1	0	0	10	10	1	1
TeacherStrategy	3	3	1	1	4	4	0,8	0,8
HomeController	1	1	0	0	6	6	1	1
ContextUserRepository	1	1	7	7	1	1	0,13	0,13
UpdatePasswordController	2	2	0	0	4	4	1	1
UserRepoStrategy	6	6	3	3	3	3	0,5	0,5
App\Models\User	2	2	6	6	1	1	0,14	0,14
App\Traits\StrategyContext	1	1	0	0	2	2	1	1
SingletonStudentStrategyRepository	-	1	-	1	-	1	-	0,5
SingletonTeacherStrategyRepository	-	1	-	1	-	1	-	0,5

Singleton pattern tidak membuat perubahan nilai metrik dikarenakan gaya penulisan kode. Pada kebanyakan kasus, pembuatan objek pada *strategy pattern* dilakukan langsung ketika konteks telah diatur. Pada langkah sebelumnya sudah ditambahkan sebuah fungsi yang diletakkan pada *traits StrategyContext* yang bertujuan membungkus pembuatan objek tersebut sehingga tidak langsung membuat objek, tetapi memanggil fungsi pada *traits* tersebut. Kendala lain dari langkah sebelumnya, pembuatan objek langsung dibuat, tidak dilakukan pemeriksaan apakah objek sebelumnya sudah ada atau belum. Berdasarkan hal tersebut, tidak ada perubahan pada metrik yang artinya tidak ada dampak spesifik secara kode, hanya menambah *class* baru.

C. Mediator Pattern

Mediator *pattern* bertujuan mengabstraksi pemanggilan berbagai *repository* ke sebuah *class mediator* sehingga mengurangi ukuran dan kompleksitas kode sehingga lebih mudah dimengerti. Hasil metrik dapat dilihat pada Tabel VIII dan Tabel IX. *Mediator pattern* memberikan dampak baik secara signifikan. Metrik *halstead volume*, LOC, WMC, dan CC untuk *class* terdampak mengalami penurunan yang signifikan sehingga nilai MI naik. Nilai *Ce* mengalami penurunan yang artinya mengurangi ketergantungan dengan *class* lain. Hal ini dikarenakan sebelumnya kode membengkak akibat pemanggilan repository yang begitu banyak sehinggameningkatkan ukuran kode dan kompleksitas, serta memengaruhi banyaknya *class* yang diperlukan sehingga menciptakan nilai ketergantungan yang tinggi.

Dengan menerapkan *mediator*, pemanggilan *repository* yang ditangani oleh *class mediator* yang menyebabkan ketergantungan *controller* terhadap *class* lain dipangkas dan hanya bergantung pada *class mediator*. Dengan begitu, *interface mediator* akan memiliki nilai *Ce* yang tinggi dikarenakan semua *class* harus diperkenalkan pada *mediator*. Sementara itu, nilai metrik *Ca* di tiap *class repository* mengalami kenaikan dikarenakan muncul *class* baru yang menggunakan mereka, yaitu *class mediator*. Perubahan nilai *Ca* dan *Ce* memengaruhi perubahan nilai *instability*. Semuanya bernilai pada rentang 0,7 hingga 1. Untuk nilai LCOM, tidak mengalami perubahan, karena hanya memindahkan dan mengganti pemanggilan *repository* ke *class mediator*.

D. Mediator vs. Template Method

Template method bertujuan mengurangi duplikasi kode. Terdapat 2 *class controller* hasil perubahan dari penerapan *mediator pattern* yang dapat diterapkan pada *template method*, yaitu *MarkController* dan *AttendanceController*. Nilai metrik *halstead volume* dan LOC mengalami penurunan dikarenakan duplikasi kode dipangkas dan diabstraksi ke sebuah fungsi di *class* baru, TemplateMethod. Penurunan pun terjadi pada nilai WMC dan CC. Meski tidak ada pengurangan fungsi, pemangkasan kode dari beberapa baris menjadi pemanggilan ke fungsi saja menjadikan kompleksitas dari fungsi tersebut menurun. Dengan menurunnya kompleksitas dan ukuran kode, maka nilai MI pun mengalami perubahan yang baik. Untuk nilai LCOM tidak mengalami perubahan, karena hanya mengurangi algoritme yang tidak berhubungan dengan varia-

TABEL VIII
HASIL METRIK SEBELUM PENERAPAN *MEDIATOR*

Class	**Metrik**									
	Halstead Volume		**LOC**		**MI**		**WMC**		**CC**	
	Bef	***Aft***	***Bef***	***Aft***	***Bef***	***Aft***	***Bef***	***Aft***	***Bef***	***Aft***
MarkController	3299,18	1988,79	185	121	54,15	65,92	28	25	22	19
UserRepository	2153,3	2153,3	118	118	53,91	53,91	26	26	16	16
AttendanceController	1049,59	404,17	94	54	74,59	89,99	15	13	10	8
PromotionController	977,37	549,84	71	52	76,79	85,74	11	11	8	8
SchoolSessionRepository	247,59	247,59	47	47	45,82	45,82	12	12	7	7
MarkRepository	761,1	761,1	46	46	42,88	42,88	11	11	5	5
SchoolClassRepository	242,03	242,03	44	44	47,05	47,05	9	9	3	3
App\Mediator\Mediator	-	90,56	-	43	-	50,53	-	2	-	1
ExamController	702,07	211,77	68	37	78,49	95,95	10	8	6	4
AcademicSettingController	505,23	122,11	55	34	74,64	90,15	6	6	3	3
CourseRepository	103,96	103,96	32	32	52,64	52,64	7	7	3	3
SectionRepository	103,96	103,96	32	32	52,64	52,64	7	7	3	3
App\Mediator\MediatorExam	-	343,87	-	30	-	49,35	-	6	-	5
AcademicSettingRepository	117,21	117,21	28	28	53,41	53,41	6	6	4	4
App\Mediator\MediatorAttendance	-	366,86	-	24	-	51,26	-	6	-	5
App\Mediator\MediatorMark	-	810,43	-	20	-	50,31	-	7	-	7
App\Mediator\MediatorPromotion	-	307,67	-	18	-	54,66	-	4	-	4
SemesterRepository	41,21	41,21	16	16	62,16	62,16	3	3	2	2
HomeController	212	16,25	27	14	94,12	115,41	2	2	1	1
App\Mediator\MediatorAcademicSetting	-	122,62	-	10	-	63,29	-	2	-	2
App\Mediator\MediatorHome	-	81,41	-	10	-	64,54	-	2	-	2

bel lokal atau fungsi lokal pada sebuah *class*. Nilai *Ca* terjadi peningkatan pada *class mediator*, karena *class* ini digunakan oleh *class* baru, yaitu *TemplateMethod*. Perbandingan nilai metrik dapat dilihat pada Tabel X dan Tabel XI.

E. Hasil Gabungan Keclima Design Pattern

Secara keseluruhan *design pattern* yang diterapkan memberikan dampak yang baik. Metrik-metrik yang menunjukkan ukuran kode dari sebuah *class*, beberapa di antaranya berkurang cukup signifikan pada nilai LOC dan *halstead volume*. *Class AttendanceController* mengalami penurunan nilai *halstead volume* mencapai 70% dan *class HomeController* mencapai 92%. Penurunan nilai LOC yang signifikan terlihat pada *class ExamController* yang mencapai 50%. Beberapa *class* menunjukkan kenaikan, namun hanya bertambah sedikit jika dibandingkan dampak positif yang

TABEL IX
HASIL METRIK SETELAH PENERAPAN *MEDIATOR*

Class	**Metrik**							
	LCOM		***Ca***		***Ce***		***Instability***	
	Bef	***Aft***	***Bef***	***Aft***	***Bef***	***Aft***	***Bef***	***Aft***
MarkController	2	2	0	0	13	4	1	1
UserRepository	10	10	0	1	10	10	1	0,91
AttendanceController	3	3	0	0	9	5	1	1
PromotionController	1	1	0	0	7	4	1	1
SchoolSessionRepository	4	4	0	1	3	3	1	0,75
MarkRepository	7	7	1	2	5	5	0,83	0,71
SchoolClassRepository	6	6	0	1	5	5	1	0,83
App\Mediator\Mediator	-	2	-	6	-	15	-	0,71
ExamController	3	3	0	0	8	5	1	1
AcademicSettingController	1	1	0	0	10	5	1	1
CourseRepository	5	5	0	1	3	3	1	0,75
SectionRepository	5	5	1	2	3	3	0,75	0,6
App\Mediator\MediatorExam	-	1	-	1	-	1	-	0,5
AcademicSettingRepository	3	3	0	1	3	3	1	0,75
App\Mediator\MediatorAttendance	-	1	-	1	-	1	-	0,5
App\Mediator\MediatorMark	-	1	-	1	-	1	-	0,5
App\Mediator\MediatorPromotion	-	1	-	1	-	1	-	0,5
SemesterRepository	2	2	0	1	3	3	1	0,75
HomeController	1	1	0	0	6	2	1	1
App\Mediator\MediatorAcademicSetting	-	1	-	1	-	1	-	0,5
App\Mediator\MediatorHome	-	1	-	1	-	1	-	0,5

TABEL X
HASIL METRIK *MEDIATOR* SEBELUM PENERAPAN *TEMPLATE METHOD*

Class	**Metrik**									
	Halstead Volume		**LOC**		**MI**		**WMC**		**CC**	
	Bef	***Aft***	***Bef***	***Aft***	***Bef***	***Aft***	***Bef***	***Aft***	***Bef***	***Aft***
MarkController	1988,79	1375,78	121	108	65,92	73,86	25	13	19	7
AttendanceController	404,17	356,62	54	51	89,99	91,75	13	11	8	6
App\Mediator\Mediator	90,56	90,56	43	43	50,53	50,53	2	2	1	1
App\Mediator\MediatorMark	810,43	803,82	20	20	50,31	50,34	7	7	7	7
TemplateMethod	-	390,14	-	40	-	45,83	-	11	-	8

TABEL XI
HASIL METRIK *MEDIATOR* SETELAH PENERAPAN *TEMPLATE METHOD*

Class	**Metrik**							
	LCOM		***Ca***		***Ce***		***Instability***	
	Bef	***Aft***	***Bef***	***Aft***	***Bef***	***Aft***	***Bef***	***Aft***
MarkController	2	2	0	0	4	4	1	1
AttendanceController	3	3	0	0	5	5	1	1
App\Mediator\Mediator	2	2	6	7	15	15	0,71	0,68
App\Mediator\MediatorMark	1	1	1	1	1	1	0,5	0,5
TemplateMethod	-	1	-	2	-	3	-	0,6

diberikan di sebagian besar *class*. *Class MarkRepository* menunjukkan kenaikan nilai *halstead volume* kurang dari 15% dan *class UserController* kurang dari 10%. Kenaikan nilai MI cukup signifikan di banyak *class*. Nilai MI yang mengalami penurunan hanya ada di 2 *class* dan tidak terlalu banyak. Secara rata-rata, nilai LOC dan *halstead volume* mengalami penurunan yang besar dan nilai MI mengalami kenaikan, meskipun tidak terlalu jauh, kurang dari 10%. Kenaikan nilai *halstead volume* mencapai 45% dan nilai LOC mencapai 31%. Artinya, *penerapan design pattern* dapat berpengaruh besar terhadap ukuran baris program dalam tingkat *class*. Pertukaran yang diperlukan untuk mencapai hal tersebut adalah menambah jumlah *class* baru yang cukup banyak, mencapai 17 *class* baru.

Kompleksitas dan tingkat kohesi dapat dinilai dari metrik WMC, CC, dan LCOM. Terlihat beberapa *class* juga memiliki dampak positif, meskipun lebih banyak *class* yang tidak mengalami perubahan. Penurunan kompleksitas yang signifikan terlihat pada nilai metrik WMC dan CC pada *class MarkController* yang mengalami penurunan mencapai 50%. Namun, ada juga yang mengalami kenaikan hingga mencapai 60% yang terjadi pada satu *class* saja. Untuk nilai LCOM, tidak ada dampak buruk yang diterima, meskipun hanya 2 *class* saja yang menunjukkan perubahan positif.

Jika dihitung dari nilai rata-rata, pengaruh *design pattern* terhadap tingkat kompleksitas menunjukkan dampak yang positif. Untuk tingkat *coupling*, dari tingkat ketergantungan sebuah *class* ke *class* lain (*Ce*) secara keseluruhan menunjukkan dampak positif, bahkan penurunan mencapai 60%. Hanya satu *class* yang mengalami kenaikan sebesar 1 nilai saja. Itupun *class model* yang jarang sekali mengalami perubahan, mengingat setiap operasi basis data dan logika bisnis sudah diatur oleh *class* lain, yaitu *controller* dan *repository*. Tingkat *Ca* banyak mengalami kenaikan. Artinya, meningkatnya modularitas dan *reusability* dari *class* tersebut, meskipun semakin banyak juga *class* yang jika mengalami perubahan, berpotensi mengganggu *class* lain yang bergantung kepadanya.

Nilai *instability* menunjukkan banyak *class* yang mengalami penurunan dan menuju ke kondisi stabil, meskipun penurunannya tidak begitu banyak. Hal ini menciptakan ketidakjelasan status pada *class* tersebut dan dapat menambah kebingungan mengenai aksi yang perlu dilakukan jika ingin melakukan perubahan atau perombakan struktur kode. Rangkuman hasil perbandingan metrik dapat dilihat pada Tabel XII dan Tabel XIII.

TABEL XII
PERBANDINGAN METRIK SEBELUM GABUNGAN *DESIGN PATTERN* DIIMPLEMENTASI

Class	**Metrik**									
	Halstead Volume		**LOC**		**MI**		**WMC**		**CC**	
	Bef	***Aft***	***Bef***	***Aft***	***Bef***	***Aft***	***Bef***	***Aft***	***Bef***	***Aft***
Builder\MarkRepositoryBuilder	-	255,33	-	53	-	45,4	-	10	-	1
AcademicSettingController	505,23	122,11	55	34	74,64	90,15	6	6	3	3
AttendanceController	1049,59	389,4	94	51	74,59	91,49	15	11	10	6
UpdatePasswordController	95,18	85,11	27	28	69,21	68,94	5	5	3	3
ExamController	702,07	211,77	68	37	78,49	95,95	10	8	6	4
HomeController	212	16,25	27	13	94,12	116,56	2	2	1	1
MarkController	3299,18	1544,05	185	111	54,15	73,2	28	13	22	7
PromotionController	977,37	534,71	71	51	76,79	86,26	11	11	8	8
UserController	1069,71	1144,21	112	123	57,56	55,45	17	17	6	6
Mediator\Mediator	-	90,56	-	43	-	66,93	-	2	-	1
Mediator\Mediator AcademicSetting	-	126,71	-	12	-	61,47	-	2	-	2
Mediator\MediatorAttendance	-	404,17	-	29	-	49,18	-	6	-	5
Mediator\MediatorExam	-	343,87	-	30	-	49,35	-	6	-	5
Mediator\MediatorHome	-	143,06	-	16	-	58,37	-	2	-	2
Mediator\MediatorMark	-	1015,45	-	42	-	42,6	-	8	-	7
Mediator\MediatorPromotion	-	299,56	-	21	-	53,28	-	4	-	4
Models\User	144,75	144,75	20	20	105,96	105,96	3	3	1	1
AcademicSettingRepository	117,21	117,21	28	28	53,41	53,41	6	6	4	4
CourseRepository	103,96	103,96	32	32	52,64	52,64	7	7	3	3
MarkRepository	761,1	832,33	46	59	42,88	62,61	11	18	5	15
SchoolClassRepository	242,03	242,03	44	44	47,05	47,05	9	9	3	3
SchoolSessionRepository	247,59	247,59	47	47	45,82	45,82	12	12	7	7
SectionRepository	103,96	103,96	32	32	52,64	52,64	7	7	3	3
SemesterRepository	41,21	41,21	16	16	62,16	62,16	3	3	2	2
SingletonStudentStrategyRepository	-	0	-	12	-	171	-	2	-	2
SingletonTeacherStrategyRepository	-	0	-	12	-	171	-	2	-	2
StudentStrategy	-	950,05	-	88	-	53,17	-	19	-	12
TeacherStrategy	-	307,71	-	41	-	46,72	-	8	-	5
Strategy\ContextUserRepository	-	63	-	29	-	55,37	-	6	-	1

Class	Metrik									
	Halstead Volume		**LOC**		**MI**		**WMC**		**CC**	
	Bef	***Aft***	***Bef***	***Aft***	***Bef***	***Aft***	***Bef***	***Aft***	***Bef***	***Aft***
Strategy\UserRepoStrategy	-	323,33	-	25	-	51,39	-	9	-	4
TemplateMethod\TemplateMethod	**-**	**394,84**	-	40	-	45,8	-	11	-	8
App\Traits\StrategyContext	-	7,92	-	15	-	67,78	-	2	-	2
Average	**600,38**	**327,81**	**55,94**	**38,15**	**63,86**	**69,45**	**9,59**	**7,38**	**5,53**	**4,32**

TABEL XIII
PERBANDINGAN METRIK SESUDAH GABUNGAN *DESIGN PATTERN* DIIMPLEMENTASI

Class	Metrik							
	LCOM		***Ca***		***Ce***		***Instability***	
	Bef	***Aft***	***Bef***	***Aft***	***Bef***	***Aft***	***Bef***	***Aft***
Builder\MarkRepositoryBuilder	-	1	-	1	-	0	-	0
AcademicSettingController	1	1	0	0	10	5	1	1
AttendanceController	3	3	0	0	9	5	1	1
UpdatePasswordController	2	2	0	0	4	4	1	1
ExamController	3	3	0	0	8	5	1	1
HomeController	1	1	0	0	6	2	1	1
MarkController	2	1	0	0	13	4	1	1
PromotionController	1	1	0	0	7	4	1	1
UserController	1	1	0	0	10	10	1	1
Mediator\Mediator	-	2	-	7	-	15	-	0,68
Mediator\MediatorAcademicSetting	-	1	-	1	-	1	-	0,5
Mediator\MediatorAttendance	-	1	-	1	-	1	-	0,5
Mediator\MediatorExam	-	1	-	1	-	1	-	0,5
Mediator\MediatorHome	-	1	-	1	-	1	-	0,5
Mediator\MediatorMark	-	1	-	1	-	1	-	0,5
Mediator\MediatorPromotion	-	1	-	1	-	1	-	0,5
Models\User	2	2	4	6	1	1	0,2	0,14
AcademicSettingRepository	3	3	0	1	3	3	1	0,75
CourseRepository	5	5	0	1	3	3	1	0,75
MarkRepository	7	4	1	2	5	6	0,83	0,75
SchoolClassRepository	6	6	0	1	5	5	1	0,83
SchoolSessionRepository	4	4	0	1	3	3	1	0,75
SectionRepository	5	5	1	2	3	3	0,75	0,6
SemesterRepository	2	2	0	1	3	3	1	0,75
SingletonStudentStrategyRepository	-	1	-	1	-	1	-	0,5
SingletonTeacherStrategyRepository	-	1	-	1	-	1	-	0,5
StudentStrategy	-	3	-	1	-	9	-	0,9
TeacherStrategy	-	3	-	1	-	4	-	0,8
Strategy\ContextUserRepository	-	1	-	3	-	1	-	0,25
Strategy\UserRepoStrategy	-	6	-	3	-	3	-	0,5
TemplateMethod\TemplateMethod	-	1	-	2	-	3	-	0,6
App\Traits\StrategyContext	-	1	-	0	-	2	-	1
Average	**3,06**	**2,26**	**0,41**	**1,29**	**5,71**	**3,44**	**0,92**	**0,69**

IV. SIMPULAN

Singleton pattern tidak memberikan dampak apapun secara metrik selain menambah jumlah *class*. *Mediator pattern* terbukti dapat mengurangi ukuran kode dan tingkat ketergantungan dengan *class* lain secara signifikan. *Template method* memberikan dampak positif dari ukuran kode dan tingkat kompleksitas program. *Strategy pattern* memberikan dampak yang tidak begitu baik dari segi ukuran program dan hampir tidak memberikan dampak pada kompleksitas, *coupling*, dan kohesi dari keseluruhan program. *Builder pattern* memberikan dampak positif untuk tingkat kohesi, namun tidak begitu baik untuk ukuran kode dan kompleksitas program. Hal ini mungkin dipengaruhi dari kasus yang diterapkan *pattern* ini, yaitu menyusun *query* ke basis data berdasarkan *filter* yang tersedia. Akibatnya menambah jumlah pemeriksaan kondisi yang berpengaruh pada peningkatan ukuran kode dan kompleksitas program. Pemeliharaan kode terpusat pada satu fungsi, namun kompleksitas naik dibandingkan versi awal yang terbagi menjadi 4 fungsi dengan masing-masing kompleksitasnya hanya bernilai 1.

Secara keseluruhan, gabungan seluruh *design pattern* yang diterapkan memberikan dampak yang baik untuk kualitas pro-

gram dari sisi pemeliharaan. Dari setiap *design pattern* yang diimplementasikan, seluruhnya membuat *class* baru, sehingga pengurangan ukuran, *coupling*, dan kompleksitas sebenarnya diabstraksi ke dalam *class* tersebut. Gaya penulisan kode dan skala program juga berdampak pada pengaruh *design pattern* terhadap pemeliharaan perangkat lunak secara metrik.

Pada penelitian ini, *design pattern* yang diterapkan berhubungan dengan ekstensi atau perubahan fitur sehingga metrik statis yang digunakan sebagai pengukuran pada penelitian ini tidak dapat menghitung dampak dari kemungkinan perubahan tersebut. Jika berhubungan dengan struktur kode, metrik statis bekerja dengan baik karena melihat kondisi dari struktur kode saat ini.

DAFTAR REFERENSI

[1] K. Elbaz dan Allaoua Chaoui, "An empirical study to improve software quality through design patterns," *Int. J. Ind. Syst. Eng.*, vol. 29, no. 1, hlm. 74–94, 2018, doi: 10.1504/IJISE.2018.091435.

[2] F. Abdullah, "Evaluating Impact of Design Patterns on Software Maintainability and Performance," University of Oslo, 2017.

[3] P. Redmond, "Laravel 9.5 Released," *Laravel News*, 2022. https://laravel-news.com/laravel-9-5-0

[4] A. Nanthaamornphong dan R. Wetprasit, "Methods to measure the impact of design patterns on software maintainability," *Maejo Int. J. Sci. Technol.*, vol. 12, no. 3, hlm. 251–271, 2018.

[5] N. Qamar dan A. A. Malik, "Impact of design patterns on software complexity and size," *Mehran Univ. Res. J. Eng. Technol.*, vol. 39, no. 2, hlm. 342–352, 2020, doi: 10.22581/muet1982.2002.10.

[6] M. Alfadel, K. Aljasser, dan M. Alshayeb, *Empirical study of the relationship between design patterns and code smells*, vol. 15, no. 4. 2020. doi: 10.1371/journal.pone.0231731.

[7] I. M. B. Gautama, S. Rochimah, dan R. J. Akbar, "Assessing the impact of enterprise software design patterns on maintainability: a case study," dalam *2019 1st International Conference on Cybernetics and Intelligent System (ICORIS)*, 2019, hlm. 128–132. doi: 10.1109/ICORIS.2019.8874885.

[8] A. Shvets, *Dive into Design Patterns*. Refactoring Guru, 2018.

[9] E. Gamma, R. Helm, R. Johnson, J. Vlissides, dan G. Booch, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1st ed. Massachusetts: Addison-Wesley Professional, 1994.

[10] N. Fenton dan J. Bieman, *Software Metrics: A Rigorous and Practical Approach*, 3rd ed. Florida: Taylor & Francis Group, 2015.

[11] R. S. Pressman, *Software Engineering: A Practitioner's Approach*, 8th ed. New York: McGraw Hill, 2014.

[12] P. Oman dan J. Hagemeister, "Construction and testing of polynomials predicting software maintainability," *J. Syst. Softw.*, vol. 24, no. 3, pp. 251–266, 1994, doi: 10.1016/0164-1212(94)90067-1.

[13] P. Redmond, "Unified Transform Open-Source School Management Platform," *Laravel News*, 2019. https://laravel-news.com/unified-transform-open-source-school-management-platform

Albertus Kevin, menerima gelar Sarjana Komputer dari Institut Teknologi Harapan Bangsa (ITHB) Bandung tahun 2022. Saat ini (2022) sedang berkarir sebagai *backend developer* di PT MEA Digital Marketing.

Hans Christian Kurniawan, menerima gelar Sarjana Teknik dari Institut Teknologi Harapan Bangsa (ITHB) Bandung tahun 2016 dan Magister Informatika dari ITB pada tahun 2019. Saat ini aktif sebagai pengajar di Prodi Informatika ITHB, juga sebagai *Engineering Manager* di Bukalapak, dan Direktur IT di PT MEA Digital Marketing.