JACIS : Journal Automation Computer Information System Volume: **4**, Nomor **2**, **2024**, *Hal*: 50 - 59

Analisis Penggunaan Pemrograman Berorientasi Obyek Terhadap *Maintainability* Perangkat Lunak Menggunakan ODOO

Muhammad Haddit Azhizi[1*], Muhammad Ainul Yaqin[2]

[1] Universitas Islam Negeri Maulana Malik Ibrahim; haditmizi@gmail.com
[2] Universitas Islam Negeri Maulana Malik Ibrahim; yaqinov@ti.uin-malang.ac.id

Abstrak: ODOO adalah framework yang populer dalam pembangunan ERP karena kemudahan konfigurasi dan adaptasinya. Penggunaan Object-Oriented Programming (OOP) dalam ODOO berpotensi meningkatkan kualitas dan pemeliharaan kode. *Maintainability* atau kemudahan pemeliharaan adalah aspek penting yang perlu diukur untuk menjamin operasional sistem ERP yang efektif. Penelitian ini mengusulkan penggunaan submetric dalam OO metric untuk mengukur *maintainability*. Pengujian dilakukan pada lima model di ODOO: ResPartner, AccountAnalyticLine, AccountAnalyticApplicability, ProductProduct, dan ProductAttributeCustomValue. Hasil pengujian menunjukkan nilai rata-rata untuk analyzability sebesar 10.4, changeability sebesar 22.4, stability sebesar 3.4, testability sebesar 3.4, dan *maintainability* sebesar 36.2. Hasil ini menandakan kelas-kelas tersebut relatif mudah dipelihara dalam jangka panjang. Namun, ada tantangan dalam analisis, perubahan, dan pengujian. Masalah utama termasuk kompleksitas warisan dan lifecycle objek, yang bisa diatasi dengan mengurangi penggunaan inheritance dan menerapkan prinsip Single Responsibility. Peningkatan dokumentasi, pengujian, dan prinsip desain diperlukan untuk meningkatkan kemudahan analisis dan stabilitas sistem

Keywords: Object Oriented Programming (OOP); ERP System; ODOO; *Maintainability*

DOI: https://doi.org/10.47134/jacis.v4i2.74
*Correspondensi: Muhammad Haddit Azhizi
Email: haditmizi@gmail.com

Receive: 27 Mei 2024
Accepted: 21 Juni 2024
Published: 27 Juni 2024

Abstrak: ODOO is a popular framework for building ERP systems due to its ease of configuration and adaptability. The use of Object-Oriented Programming (OOP) in ODOO has great potential to improve code quality and *maintainability*. *Maintainability* is a critical aspect that needs to be measured to ensure the effective operational continuity of ERP systems. This study proposes the use of submetrics in OO metrics to measure *maintainability*. Testing was conducted on five models in ODOO: ResPartner, AccountAnalyticLine, AccountAnalyticApplicability, ProductProduct, and ProductAttributeCustomValue. The results show average values for analyzability of 10.4, changeability of 22.4, stability of 3.4, testability of 3.4, and *maintainability* of 36.2. These results indicate that these *classes* are relatively easy to maintain in the long term. However, there are challenges in analysis, change, and testing. The main issues include inheritance complexity and object lifecycle complexity, which can be mitigated by reducing the use of inheritance and applying the Single Responsibility Principle. Improvements in documentation, testing, and design principles are necessary to enhance the ease of analysis and system stability.

Keywords: Object Oriented Programming (OOP); ERP System; ODOO; *Maintainability*

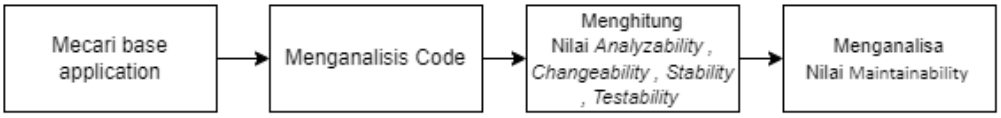
PENDAHULUAN

Sistem Perencanaan Sumber Daya Perusahaan (ERP) sangat kritikal dalam mengintegrasikan dan mengotomatisasi proses bisnis utama untuk meningkatkan efisiensi dan efektivitas operasional suatu organisasi. Sistem ERP merupakan serangkaian aplikasi bisnis atau modul yang menghubungkan berbagai unit bisnis dalam sebuah organisasi, seperti keuangan, akuntansi, produksi, dan sumber daya manusia, menjadi sebuah sistem tunggal yang terintegrasi dengan platform umum untuk arus informasi di seluruh perusahaan[1]. ODOO, sebagai salah satu platform ERP yang populer, menawarkan fleksibilitas yang signifikan karena berbasis open-source dan mendukung pengembangan yang sangat dapat disesuaikan melalui penggunaan Object-Oriented Programming (OOP). Pemrograman Berorientasi Objek (OOP) adalah paradigma pemrograman yang menggunakan "objek" – struktur data yang terdiri dari data atau atribut, dan kode dalam bentuk prosedur atau metode[2]. Dalam konteks ERP, OOP memungkinkan modifikasi dan perluasan fungsionalitas dengan cara yang terstruktur, yang sangat membantu dalam menyesuaikan sistem dengan kebutuhan bisnis yang berubah. Maintanability (kemudahan pemeliharaan) adalah aspek kritis dari perangkat lunak yang mempengaruhi waktu dan biaya yang dibutuhkan untuk membuat perubahan pada sistem, termasuk peningkatan, konfigurasi, dan debugging[3]. Dalam sistem ERP yang kompleks seperti yang dikembangkan dengan ODOO, kemampuan untuk memelihara dan memperbarui sistem dengan efisien adalah kunci untuk memastikan keberlanjutan dan efektivitas operasional jangka panjang.

Mengukur maintanability dalam proyek-proyek ERP yang dibangun dengan ODOO membantu mengidentifikasi area penting untuk peningkatan dalam desain dan implementasi. Penggunaan metrik objek-orientasi, atau OO metrics, dan submetric yang spesifik dapat memberikan wawasan yang berharga tentang aspek-aspek tertentu dari *maintainability*, seperti kemudahan dalam analisis, pengujian, dan perubahan kode. Penelitian ini bertujuan untuk mengidentifikasi dan mengukur efektivitas penggunaan OOP dalam mempengaruhi kemudahan pemeliharaan sistem ERP ODOO. Dengan menggunakan submetric dalam OO metrics, penelitian ini berusaha untuk mendapatkan pemahaman yang lebih mendalam tentang bagaimana aspek-aspek tertentu dari desain dan implementasi OOP mempengaruhi *maintainability*.

METODE

Penelitian ini dilakukan dengan tahapan yang ditunjukkan dalam gambar 1. Penelitian dimulai dengan pemilihan *base application*. *Base application* yang ditemukan nantinya akan menjadi acuan dalam penelitian ini. Kedua adalah mengalisa code yang dipakai untuk membangun aplikasi base application yang dipilih dan kemudian melakukan Analisa nilai dari *Analyzability*, *Changebility*, *Stability* dan *Testability*, Hasil analisis tersebut menjadi dasar penghitungan Maintainibility Penggunaan Pemrograman berbasi Objek (OOP) pada ODOO.



Gambar 1. Metode Penelitian

Pemilihan base application

Dalam proses pemilihan base application, analisis dilakukan terhadap berbagai aplikasi yang memenuhi kriteria spesifik yaitu Sistem ERP ODOO, ODOO dikembangkan oleh tim yang terdiri dari banyak pengembang dan telah melalui berbagai fase pengembangan, termasuk penambahan dan modifikasi fungsionalitas serta algoritma yang tercatat dalam release note. ODOO bukan hanya sebuah aplikasi Android; ini adalah suite aplikasi bisnis terintegrasi yang mencakup beragam fungsi, dari ERP (Enterprise Resource Planning) hingga CRM (Customer Relationship Management), e-commerce, manajemen persediaan, manajemen akuntansi, dan banyak lagi

Pengumpulan data

Data diperoleh dari source code ODOO yang akan dianalisa yaitu dari Modul Sales Order dengan model Respartner, AccountAnaltycyLine, AccountAnalyticApplicablity, ProductProduct, ProductAttributeCustomValue. dari beberapa model source akan dianalisa menggunakan OO metric untuk mendapatkan nilai Maintainbility

Perhitungan Nilai *Maintainability*

Aplikasi ODOO dikembangkan dengan paradigma pemrograman berbasis objek. Oleh karena itu banyak peneliti menganggap penggunaan OO metric merupakan cara yang sesuai untuk menghitung *maintainability* pada aplikasi ODOO[4]. Dalam *maintainability* terdapat 4 aspek yaitu analyzability, changeability, stability, dan testability[5]. Masing-masing aspek tersebut memiliki beberapa submetric seperti tabel 1.

Tabel 1. Submetric yang digunakan untuk menghitung OO metric

Submetric	Deskripsi	Batasan
CL_WMC	Total kompleksitas masing-masing method pada *class*	0-11
CL_CMOF	Rasio *lines of comment* dengan total lines of code	0-100
IN_BASES	Banyaknya *parent class* dalam suatu *class*	0-4
CL_FUN	Banyaknya method dalam *class*	0-9
CL_FUN PUBL	Banyaknya public method dalam *class*	0-7
CL_DATA PUBL	Banyaknya public attribute dalam *class*	0-7
CL_DATA	Banyaknya attribute dalam *class*	0-25
IN_NOC	Banyaknya child *class*	0-5
CL_STAT	Banyaknya statement yang dapat dijalankan	0-7

Dalam tabel 1, dapat diketahui bahwa masing-masing submetric memiliki batas bawah 0 serta tidak terdapat nilai negatif. Oleh karena itu dapat dikatakan bahwa semakin kecil nilainya, submetric tersebut semakin baik. Banyak statement yang dapat dijalankan dapat diartikan sebagai bagian dari code yang melaksanakan suatu aksi saat runtime ataupun compile time[6]. Nilai tersebut dapat dihitung dengan pendekatan *logical source lines of code* (logical SLOC)[7][8]. Penghitungan logical SLOC dilakukan dengan menghitung banyaknya logic statement dan statement-statement yang berkaitan[9]. Berdasarkan batasan pada tabel 1, dapat diketahui batas atas yang diterima pada analyzability, changeability, stability, testability, serta *maintainability* masing-masing adalah 121, 41, 22, 26, serta 210[10]

Menganalisa Code

Source code aplikasi dianalisis menggunakan static code analyzer Chatgpt 4.0 kemudian dibandingkan dengan batasan yang diterima pada tabel 1 di atas. Hasil penghitungan yang dituliskan dengan warna merah menunjukkan nilai di luar batas yang diterima.

Tabel 2. Perhitungan submetric pada aplikasi ODOO

model	cl_wmc	cl_cmof	in_bases	cl_fun	cl_fun_publ	cl_data_publ	cl_data	in_noc	cl_stat
ResPartner	4	6.5	1	4	4	4	4	0	30
AccountAnalyticLine	1	20	4	1	1	1	1	0	20
AccountAnalyticApplicability	0	0	4	0	0	1	1	0	15
ProductProduct	3	5	5	4	4	1	1	0	20
ProductAttributeCustomValue	0	0	5	0	0	1	1	0	10

Tabel 2 menunjukkan evaluasi *maintainability* lima model di ODOO menggunakan berbagai metrik OO. Model ResPartner memiliki kompleksitas yang tinggi dengan CL_WMC 4, CL_CMOF 6.5, dan CL_STAT 30, namun didukung dokumentasi yang baik. AccountAnalyticLine memiliki dokumentasi tinggi (CL_CMOF 20), tetapi juga kompleksitas tinggi dengan CL_STAT 20. AccountAnalyticApplicability dan ProductProduct menunjukkan kombinasi kompleksitas metode dan warisan, dengan ProductProduct memiliki CL_STAT 20. ProductAttributeCustomValue menunjukkan kompleksitas rendah dengan CL_WMC 0 dan CL_STAT 10, namun memiliki beberapa *parent class* yang bisa menjadi perhatian (IN_BASES 5). Secara keseluruhan, meskipun beberapa model menunjukkan kompleksitas tinggi, dokumentasi yang baik dan distribusi metode publik dan atribut membantu dalam *maintainability*.

Gambar 2 adalah cara menghitung nilai dari setiap parameter OO yang ada di code ODOO, menggunakan sample salah satu model yaitu ResPartner.

```python
from odoo import models, fields, api

class ResPartner(models.Model):
    _name = 'res.partner'
    _inherit = ['base']  # Inheriting from base model

    # Attributes
    name = fields.Char(string='Name')
    email = fields.Char(string='Email')
    active = fields.Boolean(string='Active', default=True)
    age = fields.Integer(string='Age')


    # Example method for Cyclomatic Complexity
    @api.multi
    def button_send_email(self):
        for record in self:
            if record.email:
                # Send an email logic would go here
                print(f"Sending email to {record.email}")
            else:
                print("No email provided.")

    @api.multi
    def update_partner_age(self, new_age):
        for record in self:
            if record.age > 0:
                record.age = new_age
            else:
                print("Invalid age provided.")

    # Static method
    @staticmethod
    def some_static_method():
        print("This is a static method.")

    # Public method
    def public_method(self):
        print("This is a public method.")
```

Gambar 2. Source Code Model ResPartner

Pada Gambar 2 merupakan source code dari model ResPartner yang ada di ODOO, yang akan menghitung nilai dari setiap parameter OO:

1. CL_WMC (*Weighted Methods per Class*)
   Untuk menghitung total *Weighted Methods per Class* (CL_WMC) pada kode di Tabel 1 menggunakan metrik *Cyclomatic Complexity* (CC), sebelumnya akan dihitung kompleksitas siklomatis masing-masing metode dalam kelas ResPartner. Kompleksitas siklomatis dihitung berdasarkan jumlah cabang kontrol (if, while, for, case, dll.) dan jumlah jalur independen dalam kode.
   Berikut adalah perhitungan CC untuk setiap metode dalam kelas ResPartner:
   a. Metode button_send_email(self):
      Terdapat 1 cabang kontrol if.
      Cyclomatic Complexity (CC) = 1 (untuk seluruh metode) + 1 (untuk pernyataan if) = 2
   b. Metode update_partner_age(self, new_age):
      Terdapat 1 cabang kontrol if.
      Cyclomatic Complexity (CC) = 1 (untuk seluruh metode) + 1 (untuk pernyataan if) = 2

   Selanjutnya akan dihitung total CL_WMC dengan menjumlahkan *Cyclomatic Complexity* dari masing-masing metode:

   CC untuk button_send_email = 2
   CC untuk update_partner_age = 2
   CL_WMC = 2+2 = 4

   Jadi, total CL_WMC untuk kelas ResPartner adalah 4

2. CL_CMOF (Rasio *lines of comment* dengan total lines of code)
   Ambil jumlah baris komentar (*lines of comment*) dan jumlah total baris kode (lines of code) dalam kelas, Hitung rasio antara jumlah baris komentar dan total baris kode.

   CL_CMOF= Total Baris Kode (TLC) / Baris Komentar (LC)
   39 / 6 = 6.5

3. IN_BASES (Banyaknya *parent class* dalam suatu *class*)
   Hitung jumlah *parent class* dari suatu *class*. Model ini hanya mewarisi dari *base*, jadi jumlahnya adalah 1.

4. CL_FUN (Banyaknya method dalam *class*)
   Hitung jumlah method dalam *class*. Dari *source* kode diatas terdapat 4 metode (*button_send_email, update_partner_age, some_static_method, public_method*)

5. CL_FUN PUBL (Banyaknya public method dalam *class*)
   Hitung jumlah method dalam *class* yang memiliki aksesibilitas publik. Semua metode kecuali *some_static_method* adalah publik. Jadi, ada 3 metode *public*.

6. CL_DATA PUBL (Banyaknya public attribute dalam *class*)
   Hitung jumlah atribut dalam *class* yang memiliki aksesibilitas publik. Semua atribut adalah publik. Jadi, ada 4 atribut publik (*name, email, active, age*).

7. CL_DATA (Banyaknya attribute dalam *class*)
   Hitung jumlah total atribut dalam *class*. Ada 4 atribut dalam kelas ini (*name, email, active, age*).

8. IN_NOC (Banyaknya child *class*)
   Hitung jumlah child *class* dari suatu *class*. Tidak ada child *class* yang ditemukan = 0

9. CL_STAT (Banyaknya statement yang dapat dijalankan)
   Hitung jumlah pernyataan (*statements*) yang dapat dijalankan dalam *class*.

Menghitung Nilai *Maintainability*

Tabel 3. Perhitungan *Maintainability*

Class	*Analyzability*	*Changebility*	*Stability*	*Testability*	*Maintainbility*
ResPartner	5	38	8	8	59
AccountAnalyticLine	25	22	2	2	51
AccountAnalyticApplicability	4	16	1	0	21
ProductProduct	13	25	5	7	50
ProductAttributeCustomValue	5	11	1	0	17
Avg	10.4	22.4	3.4	3.4	36.2

Pada Tabel 3, nilai-nilai *maintainability* diperoleh dari penjumlahan berbagai metrik yang telah dianalisis. Berikut adalah detail perhitungan dan penjelasan mengenai cara membaca nilai-nilai tersebut, persamaan untuk menghitung *maintainability* terdapat pada persamaan (1).

$$Maintainability = Analyzability + Changeability + Stability + Testability \quad ..pers\ (1)$$

1. *Analyzability* (Kemudahan di baca):

   $$Analyzability = CL\_WMC + CL\_CMOF + IN\_BASES \quad \ldots pers\ (2)$$

   CL_WMC: *Weighted Methods per Class*
   CL_CMOF: *Comment Lines of Code Fraction*
   IN_BASES: *Number of Base Classes*

2. *Changeability* (Kemudahan Perubahan):

   $$Changeability = CL\_STAT + CL\_FUN + CL\_DATA \quad \ldots pers\ (3)$$

   CL_STAT: *Number of Executable Statements*
   CL_FUN: *Number of Methods*
   CL_DATA: *Number of Attributes*

3. *Stability* (Stabilitas):

   $$Stability = CL\_DATA\_PUBL + IN\_NOC + CL\_FUN\_PUBL \quad \ldots pers(4)$$

   CL_DATA_PUBL: *Number of Public Attributes*
   IN_NOC: *Number of Child Classes*
   CL_FUN_PUBL: *Number of Public Methods*

4. *Testability* (Kemudahan Pengujian)

   Testability = CL_WMC + CL_FUN …*pers(5)*

   CL_WMC: *Weighted Methods per Class*
   CL_FUN: *Number of Methods*

Berikut adalah penghitungan pada Model ResPartner:
1. *Analyzability*: 4(CL_WMC)+0(CL_CMOF)+1(IN_BASES)=5
2. *Changeability*: 30(CL_STAT)+4(CL_FUN)+4(CL_DATA)=38
3. *Stability*: 4(CL_DATA_PUBL)+0(IN_NOC)+4(CL_FUN_PUBL)=8
4. *Testability*: 4(CL_WMC)+4(CL_FUN)=8
5. *Maintainability*: 5(Analyzability)+38(Changeability)+8(Stability)+8(Testability)=59

HASIL DAN PEMBAHASAN

Tabel 4 merupakan hasil analisis kelas dan masalah yang terkait dengan submetrik tertentu dalam konteks pemrograman berorientasi objek.

Tabel 4. Analisa Hasil *Class* terhadap Submetric

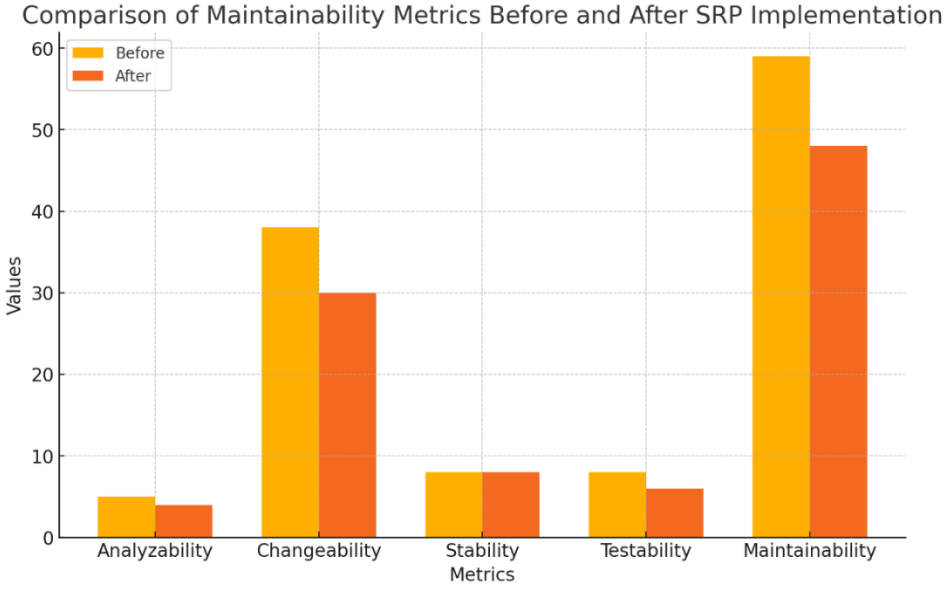
***Class* Terkait**	**Submetric yang terkait**	**Problem**
ProductProduct, ProductAttributeCustomValue	IN_BASES	Terlalu banyak *Parent class* sehingga sulit membaca *class* tersebut
Respartner, AccountAnaltycyLine, AccountAnalyticApplicablity, ProductProduct, ProductAttributeCustomValue	CL_WMC	Kompleksitas setter untuk menunjang life cycle

Berikut adalah analisis dan pembahasan dari masalah yang diidentifikasi dalam tabel 4.
1. Masalah dengan *Inheritance* (IN_BASES):
   Kelas *ProductProduct* dan *ProductAttributeCustomValue* memiliki banyak *parent class* yang membuat struktur kelas menjadi kompleks dan sulit untuk dibaca. Dalam pemrograman berorientasi objek, warisan yang berlebihan bisa menimbulkan masalah seperti kesulitan dalam memahami alur data dan perilaku dari kelas, serta potensi masalah saat melakukan modifikasi atau perluasan kode.
2. Kompleksitas dalam *Lifecycle Management* (CL_WMC):
   Kelas-kelas seperti *Respartner, AccountAnalyticLine, AccountAnalyticApplicability, ProductProduct,* dan *ProductAttributeCustomValue* menghadapi masalah dengan *setter* yang kompleks yang dibutuhkan untuk mengelola *lifecycle objek*. Kompleksitas ini mungkin mengindikasikan bahwa kelas-klas tersebut melakukan terlalu banyak tugas atau memiliki terlalu banyak tanggung jawab. Hal ini bisa menyulitkan debugging dan pemeliharaan. Untuk mengurangi kompleksitas dan meningkatkan *maintainability*, penulis memisahkan tanggung jawab berbeda ke dalam metode yang lebih kecil dan lebih fokus atau disebut dengan *Single Responsibility*. Gambar 3 adalah contoh perbaikan pada kelas *ResPartner* menggunakan *Single Responsibility*.

```python
from odoo import models, fields, api

class ResPartner(models.Model):
    _name = 'res.partner'
    _inherit = ['base']

    # Attributes
    name = fields.Char(string="Name")
    email = fields.Char(string="Email")
    active = fields.Boolean(string="Active", default=True)
    age = fields.Integer(string="Age")

    # Method to handle email sending
    @api.multi
    def button_send_email(self):
        for record in self:
            self._send_email(record)

    def _send_email(self, record):
        if record.email:
            # Send an email logic would go here
            print(f"Sending email to {record.email}")
        else:
            print("No email provided.")

    # Method to handle age update
    @api.multi
    def update_partner_age(self, new_age):
        for record in self:
            self._update_age(record, new_age)

    def _update_age(self, record, new_age):
        if record.age:
            record.age = new_age
        else:
            print("Invalid age provided.")

    # Static method
    @staticmethod
    def some_static_method():
        print("This is a static method.")

    # Public method
    def public_method(self):
        print("This is a public method.")
```

Gambar 3. Source Code Model ResPartner Menggunakan Single Responsibility

Gambar 3 menampilkan kode dari model *ResPartner* dalam ODOO yang telah diperbaiki dengan menerapkan prinsip *Single Responsibility*. Prinsip ini memastikan bahwa setiap metode dalam kelas hanya memiliki satu tanggung jawab, sehingga memudahkan analisis, pemeliharaan, dan pengujian kode.



Gambar 4 Perbandingan Perbaikan Code *Maintainability* SRP

Gambar 4 menunjukkan perbandingan metrik *maintainability* sebelum dan sesudah penerapan *Single Responsibility Principle* (SRP) pada model *ResPartner*. Grafik ini membantu untuk melihat dampak penerapan SRP terhadap beberapa aspek penting dari *maintainability* kode. Penerapan prinsip *Single Responsibility* terbukti meningkatkan beberapa aspek penting dari *maintainability* kode, yaitu:

a. *Analyzability*: Kode menjadi lebih mudah dianalisis karena tanggung jawab yang lebih jelas dan terfokus.
b. *Changeability*: Kode menjadi lebih mudah diubah dengan dampak yang lebih kecil pada bagian lain dari sistem.
c. *Testability*: Kode menjadi lebih mudah diuji karena metode yang lebih kecil dan lebih spesifik.

Secara keseluruhan, penerapan prinsip desain yang baik seperti *Single Responsibility Principle* dapat secara signifikan meningkatkan kualitas dan *maintainability* kode dalam jangka panjang, membuat proses pengembangan dan pemeliharaan menjadi lebih efisien dan andal.

SIMPULAN

Nilai rata-rata kemudahan pemeliharaan pada ODOO sebesar 36.2 menunjukkan bahwa secara umum, sistem memiliki tingkat pemeliharaan yang cukup tinggi. Namun, dari masalah yang terungkap, terdapat area spesifik yang memerlukan perhatian untuk lebih meningkatkan kemudahan pemeliharaan tersebut seperti kompleksitas warisan (*Inheritance Complexity*) dimana masalah dengan banyaknya *parent class* yang menyebabkan kesulitan dalam membaca dan memahami kelas menunjukkan bahwa sistem memiliki struktur warisan yang mungkin terlalu kompleks. Ini dapat mempersulit pemeliharaan karena perubahan pada satu kelas bisa memiliki efek tak terduga pada kelas lain yang mewarisinya. Pengurangan kompleksitas ini, mungkin dengan cara mengurangi penggunaan inheritance dan menggantinya dengan komposisi, dapat membantu meningkatkan kemudahan pemeliharaan. Selain itu kompleksitas *lifecycle* juga dengan masalah kompleksitas *setter* untuk menunjang *lifecycle* menunjukkan bahwa terdapat *overhead* dalam pengelolaan *state* dan *lifecycle objek*. Ini menunjukkan bahwa beberapa kelas mungkin terbebani dengan tanggung jawab yang terlalu banyak, menyulitkan untuk pemeliharaan dan modifikasi. Memisahkan tanggung jawab melalui penerapan prinsip *Single Responsibility* dan mungkin memperkenalkan pola desain yang lebih cocok untuk mengelola *lifecycle* dapat membantu.

DAFTAR PUSTAKA

[1] M. Muhaimin and L. Bachtiar, "Pengembangan Teknologi ERP Untuk Modul Human Resource Management Studi Kasus PT Imperium Happy Puppy Sampit," *J. Ilm. Betrik*, vol. 2, no. 1, pp. 71–78, 2020, doi: 10.36050/betrik.v11i3.207.

[2] L. E. Zen and D. U. Iswavigra, "Critical Review: Analogi RAD, OOP dan EUD Method dalam Proses Development Sistem Informasi," *J. Inf. dan Teknol.*, vol. 5, no. 1, pp. 184–190, 2023, doi: 10.37034/jidt.v5i1.286.

[3] M. Adhy, B. Priyambadha, and F. Pradana, "Kuantifikasi Pengaruh Understandability Dan *Maintainability* Pada Evolusi Perangkat Lunak," *J. Teknol. Inf. dan Ilmu Komput.*, vol. 6, no. 3, pp. 229–234, 2019, doi: 10.25126/jtiik.201961289.

[4] A. D. F. Martins, C. Melo, J. M. Monteiro, and J. de Castro Machado, "Empirical study

about *class* change proneness prediction using software metrics and code smells," *ICEIS 2020 - Proc. 22nd Int. Conf. Enterp. Inf. Syst.*, vol. 1, no. Iceis, pp. 140–147, 2020, doi: 10.5220/0009410601400147.

[5] L. M. Andrä, B. Taufner, S. Schefer-Wenzl, and I. Miladinovic, "*Maintainability* Metrics for Android Applications in Kotlin: An Evaluation of Tools," *ACM Int. Conf. Proceeding Ser.*, pp. 51–59, 2020, doi: 10.1145/3393822.3432334.

[6] H. H. Elmousalami, "Artificial Intelligence and Parametric Construction Cost Estimate Modeling: State-of-the-Art Review," *J. Constr. Eng. Manag.*, vol. 146, no. 1, 2020, doi: 10.1061/(asce)co.1943-7862.0001678.

[7] A. Darmawan, R. Ratnadewi, and A. Prijono, "Designing a wireless robot plotter as a supporting tool for understanding logical thinking," *J. Mantik*, vol. 7, no. 1, pp. 155–167, 2023, doi: 10.35335/mantik.v7i1.3640.

[8] N. C. Setiyono, B. N. Satria, and S. R. Wicaksono, "Pengukuran Sistem Informasi Menggunakan Metode Function Point Analysis (Studi Kasus: Software FrontAccounting)," *J. Janitra Inform. dan Sist. Inf.*, vol. 1, no. 2, pp. 115–123, 2021, doi: 10.25008/janitra.v1i2.139.

[9] P. Trojovský and M. Dehghani, "Pelican Optimization Algorithm: A Novel Nature-Inspired Algorithm for Engineering Applications," *Sensors*, vol. 22, no. 3, 2022, doi: 10.3390/s22030855.

[10] G. Bagus, V. Putraadinatha, D. Dwi, J. Suwawi, and S. Y. Puspitasari, "Pengaruh Design Pattern Terhadap *Maintainability* Aplikasi Mobile," *e-proceeding Eng.*, vol. 8, no. 5, p. 10954, 2021.