JITET (Jurnal Informatika dan Teknik Elektro Terapan)
Vol. 13 No. 3, pISSN: 2303-0577 eISSN: 2830-7062 http://dx.doi.org/10.23960/jitet.v13i3.6817

PEMBANGUNAN MODEL DATA SKALA BESAR DENGAN MENGGUNAKAN BUILDER DESIGN PATTERN

Shelma Aulia Maharani[1*], Putri Fisichella Chasannove[2], Carudin[3]

[1,2]Universitas Singaperbangsa Karawang; Jl. HS. Ronggo Waluyo, Puseurjaya, Telukjambe Timur, Karawang, Jawa Barat 41361; Telp. (0267) 64177

Keywords:
Builder Design Pattern;
Large-scale Data Modeling;
Python;
Google Colab;
Software Architecture

Corespondent Email:
shelforworking@gmail.com

Abstrak. Penelitian ini membahas pembangunan model data skala besar menggunakan pendekatan Builder Design Pattern pada lingkungan pemrograman Python di Google Colab. Permasalahan dalam pengelolaan dan konstruksi data berukuran besar yang bersifat dinamis seringkali menyebabkan kode program menjadi tidak terstruktur dan sulit dipelihara. Penelitian ini bertujuan membangun kerangka modular untuk membentuk model data yang fleksibel dan dapat diperluas menggunakan pola desain builder. Metode penelitian mencakup perancangan arsitektur perangkat lunak, implementasi kode dengan pendekatan builder pattern, dan evaluasi performa serta struktur kode. Hasil menunjukkan bahwa pendekatan ini menghasilkan struktur kode yang lebih terorganisir dan mudah untuk dikembangkan lebih lanjut. Kesimpulannya, penggunaan Builder Design Pattern dapat meningkatkan maintainability dan scalability dalam pengembangan model data skala besar.

Abstract. *This research discusses the construction of large-scale data models using the Builder Design Pattern approach in Python programming on the Google Colab platform. The challenges in managing large and dynamic datasets often lead to unstructured and unmaintainable code. This study aims to build a modular framework to construct flexible and extensible data models using the builder design pattern. The research method includes software architecture design, code implementation with the builder pattern approach, and performance and code structure evaluation. The results show that this approach produces more organized and extensible code structures. In conclusion, the use of the Builder Design Pattern improves maintainability and scalability in the development of large-scale data models.*

1. PENDAHULUAN

Transformasi digital yang pesat di berbagai sektor industri telah mendorong peningkatan volume, variasi, dan kecepatan data yang dihasilkan. Hal ini menuntut sistem perangkat lunak untuk dapat mengelola model data dalam skala besar secara efisien dan terstruktur. Model data skala besar merupakan representasi kompleks dari entitas dunia nyata yang terdiri dari ribuan atribut, relasi, dan konfigurasi yang saling terkait dan seringkali berubah secara dinamis [1].

Dalam praktik pengembangan perangkat lunak, terutama dengan bahasa pemrograman *Python*, model data sering kali dikonstruksi secara langsung menggunakan struktur data dasar seperti dictionary, list, atau class sederhana [5]. Pendekatan ini memiliki keterbatasan, terutama dalam hal skalabilitas dan maintainability. Ketika struktur data bertambah kompleks, kode program cenderung menjadi monolitik, sulit dimodifikasi, dan rentan terhadap kesalahan. Permasalahan ini menjadi lebih signifikan ketika pengembangan dilakukan oleh tim besar atau ketika model data digunakan secara berulang dalam berbagai konteks.

Salah satu pendekatan yang dapat digunakan untuk mengatasi permasalahan ini adalah penerapan *Builder Design Pattern.* Pola desain ini merupakan bagian dari kelompok *creational pattern* yang bertujuan untuk memisahkan proses konstruksi objek kompleks dari representasi akhirnya. Dengan demikian, pola ini memungkinkan pembangunan objek dilakukan secara bertahap dan modular [4]. Dalam konteks model data, *Builder Pattern* memberikan fleksibilitas dalam penambahan fitur baru, validasi data selama proses pembangunan, serta pengaturan dependensi antar atribut data secara lebih terorganisir.

Beberapa penelitian terdahulu telah membahas penggunaan *Builder Pattern* dalam berbagai konteks, seperti pengembangan antarmuka pengguna, dan pengelolaan konfigurasi system [2][3]. Namun, penerapan khususnya dalam pembangunan model data skala besar, terutama dengan konteks *Python* dan ekosistem *Google Colab,* masih terbatas. Padahal, *Google Colab* merupakan platform berbasis cloud yang sangat populer dalam dunia pendidikan dan penelitian karena kemampuannya menjalankan kode *Python* secara interaktif, mengakses *resource* komputasi tinggi, serta kolaborasi waktu nyata antar pengguna.

Adapun tujuan utama dari penelitian ini adalah:

1. Mendesain arsitektur perangkat lunak berbasis *Builder Design Pattern* yang mendukung pembangunan model data skala besar.
2. Mengimplementasikan arsitektur tersebut dalam bahasa *Python* pada platform *Google Colab.*

Dengan pendekatan ini, diharapkan hasil penelitian dapat memberikan kontribusi nyata terhadap praktik pengembangan perangkat lunak berbasis data serta menambah referensi implementatif pola desain dalam konteks pemrograman *Python* modern.

2. TINJAUAN PUSTAKA

2.1 Builder Design Pattern

Builder Design Pattern merupakan salah satu dari creational design pattern yang fokus pada proses pembuatan objek kompleks dengan menyediakan antarmuka untuk membuat bagian-bagian dari objek tersebut secara bertahap [3]. Pola ini memisahkan pembuatan objek dari representasinya, sehingga kode menjadi lebih modular dan fleksibel terhadap perubahan struktur data.

2.2 Model Data Skala Besar

Model data skala besar sering digunakan dalam aplikasi analitik, pembelajaran mesin, dan sistem basis data modern. Struktur model ini bisa mencakup ribuan atribut dan memerlukan pendekatan sistematis dalam pembangunannya [6].

2.3 Python dan *Google Colab*

Python merupakan bahasa pemrograman populer dalam bidang sains data dan rekayasa perangkat lunak. *Google Colab* adalah lingkungan berbasis *cloud* yang mendukung *Python* dan memungkinkan eksekusi kode secara interaktif dengan kapasitas pemrosesan yang besar [7].

3. METODE PENELITIAN

3.1 Rancangan Penelitian

Penelitian ini bertujuan untuk mengembangkan dan menguji efektivitas penggunaan *Builder Design Pattern* dalam proses konstruksi model data skala besar menggunakan bahasa pemrograman *Python*. Jenis penelitian yang digunakan adalah pengembangan *(developmental research)* karena menghasilkan suatu artefak berupa arsitektur dan kode program *builder* yang dapat digunakan untuk membentuk model data dalam skala besar. Penelitian dilakukan dalam lingkungan *Google Colab* untuk mendukung kebutuhan komputasi dan kemudahan kolaborasi eksperimen

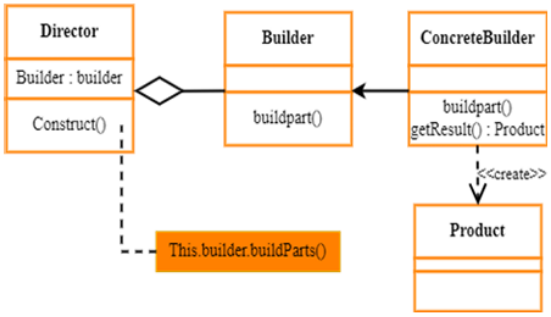
3.2 Desain Arsitektur Sistem

Tahap awal penelitian adalah perancangan arsitektur perangkat lunak dengan menerapkan prinsip dari *Builder Design Pattern*. Komponen utama dalam arsitektur ini terdiri dari :

- *Director*: Bertugas mengatur urutan dan logika konstruksi model data.
- *Abstract Builder: Interface* yang mendefinisikan langkah-langkah generik pembuatan objek model.
- *Concrete Builder (LargeDataModelBuilder):* Implementasi dari abstract *builder* yang

merealisasikan setiap langkah sesuai dengan kebutuhan model data.

- *Product* (Model Data): Objek akhir yang dihasilkan dari proses konstruksi *builder*.



Gambar 1. Diagram UML *Builder Design Pattern*

Struktur arsitektur ini memungkinkan pemisahan yang jelas antara cara membangun objek dan representasi objek yang dihasilkan, sehingga memungkinkan fleksibilitas dan ekspansi atribut model di masa mendatang [8].

3.4 Implementasi dalam *Python* di *Google Colab*

Tahap implementasi dilakukan dalam lingkungan Google Colab menggunakan bahasa pemrograman Python. Pemilihan Google Colab dilatarbelakangi oleh kemampuan eksekusi komputasi awan, dukungan terhadap pustaka pihak ketiga, serta kemudahan visualisasi hasil eksperimen.

Kode implementasi dibuat dalam bentuk modular menggunakan pendekatan OOP *(Object-Oriented Programming)*. Setiap komponen dari pola *builder* diimplementasikan sebagai kelas *Python* yang dapat diinisialisasi, diuji, dan dikembangkan secara terpisah. Berikut kode program Implementasi *Builder Pattern* untuk model data :

```
# Builder Interface
class ModelBuilder:
    def reset(self):
        pass

    def set_feature_a(self, value):
        pass

    def set_feature_b(self, value):
        pass
    def get_result(self):
        pass

# Concrete Builder
class LargeDataModelBuilder(ModelBuilder):
    def __init__(self):
        self.reset()

    def reset(self):
        self.model = {}

    def set_feature_a(self, value):
        self.model["feature_a"] = value

    def set_feature_b(self, value):
        self.model["feature_b"] = value

    def get_result(self):
        return self.model

# Director
class Director:
    def __init__(self, builder):
        self._builder = builder

    def construct_minimal_model(self):
        self._builder.reset()
        self._builder.set_feature_a("Basic")

    def construct_full_model(self):
        self._builder.reset()
        self._builder.set_feature_a("Advanced")
        self._builder.set_feature_b("High
Capacity")

# Client Code in Google Colab
builder = LargeDataModelBuilder()
director = Director(builder)

# Build full model
director.construct_full_model()
model_data = builder.get_result()
print(model_data)
```

Tabel 1. Kode Program Implementasi *Builder Pattern*

3.5 Teknik Evaluasi

Evaluasi dilakukan dengan membandingkan kompleksitas kode, waktu eksekusi, dan fleksibilitas perubahan struktur model terhadap pendekatan non-builder (konvensional).

4. HASIL DAN PEMBAHASAN

4.1 Hasil Implementasi

Penerapan *Builder Design Pattern* di *Google Colab* menghasilkan struktur kode *Python* yang modular dan dapat diperluas. Model data yang

dibentuk menggunakan LargeDataModelBuilder lebih mudah untuk dikembangkan dibanding pendekatan konvensional yang langsung membentuk struktur data dalam satu blok kode. Output kode program :

Gambar 2. Output Hasil Implementasi Model Data *Builder Design Pattern*

Selain itu, waktu pengembangan dan pengujian menjadi lebih efisien karena fungsi-fungsi pembangunan objek dipisah secara eksplisit dalam class builder dan director.

4.2 Perbandingan dengan Pendekatan Konvensional

Untuk menilai efektivitas penerapan *Builder Design Pattern* dalam membangun model data skala besar, dilakukan perbandingan langsung dengan pendekatan konvensional. Pendekatan konvensional umumnya dilakukan dengan menyusun seluruh atribut data secara langsung dalam satu blok kode atau fungsi. Hal ini terlihat sederhana untuk model kecil, tetapi menjadi cepat kompleks dan sulit dipelihara ketika jumlah atribut bertambah dan struktur data menjadi bersarang.

4.2.1 Pendekatan Konvensional

Dalam pendekatan konvensional, setiap perubahan struktur (misalnya menambah field baru atau mengubah validasi) harus dilakukan langsung di bagian kode tersebut, yang meningkatkan risiko error dan mengurangi keterbacaan kode.

```
def create_data_model():
model = {
"id": "user001",
"feature_a": "Advanced",
"feature_b": "High Capacity",
"created_at": "2025-06-01T08:00:00",
"region": "Asia",
"settings": {
"theme": "dark",
"language": "en",
"notifications": True
}
}
return model
```

Tabel 2. Kode Program Pendekatan Konvensional

4.2.2 Pendekatan *Builder*

Builder Design Pattern mengorganisasi proses pembentukan objek dalam metode-metode terpisah yang dapat dikombinasikan dan dipanggil sesuai kebutuhan.

```
builder = LargeDataModelBuilder()
builder.set_id("user001")
builder.set_feature_a("Advanced")
builder.set_feature_b("High Capacity")
builder.set_region("Asia")
builder.set_settings(theme="dark",
language="en", notifications=True)
model = builder.get_result()
```

Tabel 3. Kode Program Pendekatan *Builder*

4.3 Evaluasi Hasil Implementasi

Kriteria	Builder Pattern	Konvensional
Struktur Kode	Modular, terorganisir	Monolitik, tidak terpisah
Ekstensibilitas	Tinggi	Rendah
Maintainability	Tinggi	Sulit Dipelihara
Kemudahan Refactoring	Mudah	Kompleks

Tabel 4. Evaluasi Hasil Implementasi

- **Struktur Kode dan Modularitas**

Builder menghasilkan struktur kode yang modular dan reusable. Metode-metode seperti set_feature_a dan set_settings dapat digunakan kembali di berbagai konteks atau disesuaikan untuk membentuk variasi model. Sedangkan Pendekatan konvensional cenderung menghasilkan kode terpusat (monolitik) yang sulit dipecah atau diuji secara unit.

- **Ekstensibilitas**

Builder pattern mudah dikembangkan: cukup menambahkan metode baru pada class builder. Misalnya, penambahan atribut baru seperti set_last_login() tidak akan mengganggu kode yang telah ada. Sebaliknya, pendekatan konvensional memerlukan modifikasi di seluruh lokasi yang menggunakan struktur model tersebut.

- ***Maintainability* dan *Testing***

Karena *builder* memisahkan logika pembentukan objek, pengujian unit (unit *testing*) dapat dilakukan terhadap setiap metode *builder* secara terpisah. Hal ini tidak mudah dilakukan dalam pendekatan konvensional karena logika pembentukan objek bersatu dengan pemrosesan lainnya dalam satu fungsi.

Dengan demikian, dapat disimpulkan bahwa meskipun pendekatan builder memerlukan perancangan awal yang lebih kompleks dan jumlah baris kode sedikit lebih banyak, keunggulannya dalam hal struktur, fleksibilitas, dan efisiensi pengembangan menjadikannya pilihan yang lebih baik untuk proyek yang berskala menengah hingga besar dan bersifat dinamis.

5. KESIMPULAN

Penelitian ini menunjukkan bahwa :

1. Builder Design Pattern menghasilkan struktur kode yang modular dan terorganisir, sehingga mempermudah pengembangan, pemeliharaan, serta ekspansi struktur data dalam proyek berskala besar dan kompleks.
2. Pendekatan builder memberikan fleksibilitas tinggi dalam konstruksi model data, karena setiap komponen model dibangun melalui metode terpisah yang dapat digunakan kembali dan dimodifikasi secara independen.
3. Perbandingan dengan pendekatan konvensional menunjukkan bahwa builder unggul dalam hal keterbacaan kode, maintainability, kemudahan refactoring, serta pengurangan kompleksitas kognitif yang signifikan.
4. Google Colab terbukti sebagai platform yang cocok untuk penerapan dan eksperimen Builder Pattern karena mendukung eksekusi Python interaktif, kolaborasi real-time, serta integrasi pustaka yang dibutuhkan untuk pemrosesan data berskala besar.

UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada pihak-pihak terkait yang telah memberi dukungan terhadap penelitian ini. Terima kasih kepada diri saya sendiri, orang tua yang selalu mendukung saya, dosen pengampu mata kuliah, Adisty dan Gang Wisuki yang selalu menjadi alasan saya semangat dalam menjalani kehidupan. Lalu teman teman kuliah saya yang telah menemani saya selama proses pengembangan diri di kehidupan kuliah ini.

DAFTAR PUSTAKA

[1] Iqbal Ramadhani Mukhlis, BIG DATA Mengenal Big Data & Implementasinya di Berbagai Bidang. 2024.

[2] E. Gamma, R. Helm, R. Johnson, dan J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[3] L. Bass, P. Clements, dan R. Kazman, *Software Architecture in Practice*, 2nd ed. Reading, MA: Addison Wesley, 2003.

[4] Albertus Kevin, Analisis Pengaruh Design Pattern Terhadap Pemeliharaan Perangkat Lunak Learning Management System, 2023.

[5] Mambang, Finki Dona Marleny, Algoritma Pemrograman Menggunakan *Python*, 2022.

[6] D. D. Putri, G. F. Nama, dan W. E. Sulistiono, "Analisis Sentimen Kinerja Dewan Perwakilan Rakyat (DPR) Pada Twitter Menggunakan Metode Naive Bayes Classifier," *Jurnal Informatika dan Teknik Elektro Terapan*, vol. 10, no. 1, 2022.

[7] Google, "Welcome to Colaboratory," [Online]. Tersedia : https://colab.research.google.com/

[8] Nabila Khairunisa, Pengaruh Builder Design Pattern Pada Performa Aplikasi Dengan Menggunakan Bahasa Pemrograman Golang, Majalah Ilmiah UNIKOM, 2024.