UDC: 004.415.2:005.1

OBJECT-ORIENTED DESIGN QUALITY IN SERVICE-BASED SYSTEMS: A CASE STUDY OF A JAVA CAR REPAIR MANAGEMENT SYSTEM

Maftunaxon G'ulomova O'tkirjon qizi
Information Technology Student, Millat Umidi University
E-mail: gulomova.m1209@gmail.com
Scientific Supervisor:
***Muslimbek Pirnazarov** PhD in Computer Science*

Annotatsiya: Ushbu maqolada Java dasturlash tilida ishlab chiqilgan avtomobil ta'mirlash boshqaruv tizimi asosida obyektga yo'naltirilgan dasturlash (OOP) dizaynining dasturiy sifat ko'rsatkichlariga ta'siri tahlil qilingan. Tadqiqot modullik, saqlanish va kengaytirilish kabi mezonlarga asoslangan. Tizim Driver, Service, Booking va AutoService kabi real obyektlarni modellashtiradi va CarRepairing boshqaruv klassi orqali ishlaydi. Natijalar OOP dizayn tizim murakkabligini kamaytirishini va kengaytirishni osonlashtirishini ko'rsatadi.

Kalit so'zlar: Java, OOP, tizim dizayni, modullik, arxitektura

Аннотация: В статье анализируется влияние объектно-ориентированного проектирования на качество программных систем на основе Java-системы управления автосервисом. Рассматриваются модульность, сопровождаемость и расширяемость системы. Результаты показывают, что ООП снижает сложность и улучшает структуру системы.

Ключевые слова: Java, ООП, архитектура системы, модульность

Abstract: This paper analyzes the impact of object-oriented design on software quality using a Java-based Car Repair Management System. The study focuses on modularity, maintainability, and extensibility. The system models real-world entities such as drivers, services, and bookings under a centralized architecture. The findings show that object-oriented design reduces system complexity and improves structural organization.

Keywords: Java, Object-Oriented Programming, Software Design, Modularity, Maintainability

Introduction

In the context of modern software engineering, the increasing complexity of real-world systems requires structured and scalable design approaches. Service-based systems, such as car repair management platforms, involve multiple interacting entities and continuous data flow. Without proper architectural organization, such systems tend to become difficult to maintain and extend.

Object-oriented programming provides a conceptual and practical framework for managing this complexity. By representing real-world elements as objects and defining clear relationships between them, OOP enables developers to decompose systems into manageable components. However, the effectiveness of object-oriented design depends on how well its principles are applied in practice.

This study examines the impact of object-oriented design decisions on software quality using a Java-based Car Repair Management System as a case study. The research focuses on how modular structure, encapsulation, and data management strategies influence system maintainability and extensibility.

Literature Review

Object-oriented programming has been extensively studied as a fundamental paradigm in software engineering. According to established research, OOP enhances system modularity through abstraction, encapsulation, and separation of concerns. These characteristics allow developers to manage complexity by dividing systems into smaller, independent components.

Java, as a fully object-oriented programming language, provides a rich environment for implementing such designs. Its class-based structure and built-in collection framework support efficient data organization and manipulation. In particular, structures such as HashMap allow fast access and management of system entities, which is essential in dynamic service systems.

Previous studies indicate that modular design reduces interdependency between system components and improves long-term maintainability. At the same time, low coupling and high cohesion are identified as key indicators of software quality. Despite these findings, there remains a need for applied case studies that demonstrate how these principles function within real system models.

Methodology

The research adopts a case study approach in which a Car Repair Management System is designed and analyzed using object-oriented principles. The system models key real-world entities, including drivers, services, bookings, and service centers. Each entity is represented as a separate class, reflecting the principle of modular decomposition.

The overall system is managed through a central controller class, CarRepairing, which coordinates operations and maintains system state. Data is stored using Java’s HashMap structure, allowing efficient mapping between unique identifiers and system objects.

The methodological approach focuses on structural analysis rather than performance benchmarking. The system is evaluated based on its design characteristics, including modularity, maintainability, extensibility, and coupling between components.

System Design and Implementation

The architecture of the system follows a centralized control model combined with distributed object representation. Each class encapsulates specific data and behavior, ensuring that responsibilities are clearly separated across the system.

Encapsulation is implemented by restricting direct access to class attributes and providing controlled methods for interaction. For example, the Driver class stores user information while exposing only necessary access methods:

```java
class Driver {
    private int id;
    private String name;

    public Driver(int id, String name) {
        this.id = id;
        this.name = name;
    }

    public int getId() {
        return id;
    }
}
```

This approach protects internal data and reduces unintended dependencies between components.

In addition to encapsulation, efficient data management plays a crucial role in system performance. The system employs Java's HashMap collection to store and retrieve entities based on unique identifiers:

```java
Map<Integer, Driver> drivers = new HashMap<>();
```

The use of HashMap allows constant-time average complexity for retrieval operations, making it suitable for managing dynamic system data where frequent access is required.

Furthermore, object interaction within the system is modeled through relationships between classes. For example, the Booking class connects different system entities, representing a real-world service transaction:

```java
class Booking {
    private Driver driver;
    private Service service;

    public Booking(Driver driver, Service service) {
        this.driver = driver;
        this.service = service;
    }
}
```

This structure reflects real-world interactions and enhances the logical clarity of the system design.

Results and Discussion

The analysis of the implemented system demonstrates that object-oriented design significantly improves structural organization and reduces system complexity. By decomposing the system into independent classes, the design minimizes coupling and enhances modularity.

The effectiveness of encapsulation, as demonstrated in the Driver class implementation, ensures that system data remains protected and accessible only through controlled interfaces. Similarly, the use of HashMap for data storage provides efficient handling of system entities, contributing to improved performance.

The interaction model implemented through the Booking class illustrates how object relationships can represent real-world processes. This approach not only improves readability but also makes the system easier to extend, as new relationships can be introduced without restructuring existing components.

These findings confirm that combining object-oriented principles with appropriate data structures leads to a well-organized and scalable system.

Conclusion

This research examined the role of object-oriented design in improving software quality through a Java-based Car Repair Management System. The findings indicate that modular decomposition, encapsulation, and appropriate data management significantly enhance maintainability and extensibility.

The results support the conclusion that object-oriented programming is an effective methodology for designing scalable and structured service-based systems. Its application allows developers to manage complexity more effectively and build systems that are easier to maintain and expand over time.

References:

1. Booch G. Object-Oriented Analysis and Design // Addison-Wesley, 1994.
2. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software // Addison-Wesley, 1995.
3. Eckel B. Thinking in Java // Prentice Hall, 2006.
4. Oracle Corporation. Java Documentation // Oracle, 2023.
5. Gosling J., Joy B., Steele G., Bracha G. The Java Language Specification // Addison-Wesley, 2014.