XI’AN JIAOTONG-LIVERPOOL UNIVERSITY 西 交 利物 浦 大 学

COURSEWORK SUBMISSION COVER Page

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Group Number |  | | | | | |
| Students' ID Number |  |  |  |  |  |  |
| Module Code |  | | | | | |
| Assignment Title |  | | | | | |
| Submission Deadline |  | | | | | |

By uploading this coursework submission with this cover page, we certify the following:

 We have read and understood the definitions of collusion, copying, plagiarism, and dishonest use of data as outlined in the Academic Integrity Policy of Xi’an Jiaotong- Liverpool University.

 This work is **entirely** our own, original work produced specifically for this assignment. It does not misrepresent the work of another group or institution as our own.

 This work is not the product of unauthorized collaboration between ourselves and others.

 This work has not been shared wholly or in part outside of the group.

 It is a submission that has not been previously published, or submitted to any modules.

 Students who would like to submit the same or similar work from previous years to the current module or other modules must receive written permission from all instructors involved in advance of the assignment due date.

 **All** group members are **equally** and **collectively** responsible for the **entire** submission. Violations of academic integrity including failure to monitor group member contribution originality constitutes negligence.

 Unreported suspicious academic misconduct by one group member will be attributed to ALL members in terms of penalties. **ALL members share the responsibilities**.

 Use of generative AI is strictly prohibited for all assessments involved in this module.

We understand collusion, plagiarism, dishonest use of data, and submission of procured work are serious academic misconducts. **All** group members are held **jointly accountable** for the integrity

of the **entire** submission. By uploading or submitting this cover page, we acknowledge that we are jointly subject to penalties and disciplinary actions if we are found to have committed such acts.

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ We acknowledge that the university late submission policy will be applied if applicable.

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Please list the ID number of any group member NOT contributing to the submission:

Please indicate whether based on your individual contributions you meet the learning outcomes covered by this coursework and confirm your submission meets with all above requirements by signing your name, handwritten in ink, in English (Foreign students) or PINYIN (local students):

|  |  |  |  |
| --- | --- | --- | --- |
| **Student ID** | **(tick) box to confirm you contributed to all cw tasks** | **(tick) box to confirm you meet all LOs covered by this cw** | **Signed** |
|  | □ | □ |  |
|  | □ | □ |  |
|  | □ | □ |  |
|  | □ | □ |  |

Date:

1. **Introduction**

In the history of software engineering, engineers have constantly searched for a method that could completely solve the challenges of software development. However, in 1986, renowned computer scientist Fred Brooks presented a groundbreaking view in his paper *"No Silver Bullet: Essence and Accidents of Software Engineering"*: there will be no single technology or method capable of increasing software development productivity by an order of magnitude.

A "silver bullet" refers to a mythical weapon that can instantly eliminate all problems with a single shot. The idea of "no silver bullet" means that in software development, there is no magic solution that can instantly remove all difficulties or dramatically improve efficiency. Fred Brooks argued that the greatest challenges in software development stem from the inherent complexity of the problems themselves, not from the limitations of implementation tools. This viewpoint has remained highly influential and is widely regarded as a foundational theory for understanding the true nature of software engineering.

1. **Identified Key Challenges**

In "No Silver Bullet: Essence and Accidents of Software Engineering", Fred Brooks divides the difficulties of software engineering into two categories: essential and accidental. He particularly emphasizes the essential challenges, as these cannot be completely eliminated through technological means. The following are the key challenges in software development discussed in the article:

1. **Complexity**

Software systems are inherently complex because they need to handle a wide variety of states, inputs, logic, and interactions. Each module is highly interconnected, and modifying one part may impact the entire system. Moreover, unlike other engineering products that often have repetitive structures, software design is usually customized, with no universal "template" to rely on. As the system grows in scale, this complexity increases exponentially, significantly raising the difficulty of development, testing, and maintenance.

1. **Conformity**

Software must adapt to various external environments and constraints, such as laws and regulations, business processes, user requirements, and hardware platforms. These external demands are often complex and even conflicting, which means the software must continuously adjust itself to maintain conformity. Because these requirements come from multiple sources, it's nearly impossible to establish a single, unified principle to simplify them. As a result, developers are faced with the difficult task of constantly adapting the software to meet these diverse expectations.

1. **Changeability**

Unlike constructing a building with a fixed "design blueprint," software does not have a fixed "design blueprint" and is highly variable. As a result, there are high expectations for its adaptability. In reality, software systems need to be continuously modified to accommodate changes in user requirements, market conditions, and even management decisions. However, frequent changes can introduce new errors, disrupt the existing structure, and increase maintenance costs. Therefore, balancing the need for change while maintaining the stability of the system is one of the major challenges in software engineering.

1. **Invisibility**

Software lacks a physical form, and unlike constructing buildings, software developers cannot clearly understand the structure of the software through a blueprint like they would for a house. This invisibility makes software design difficult to comprehend, leading to misunderstandings in communication among developers, which in turn affects development efficiency and team collaboration. Moreover, the lack of an intuitive representation can lead to information asymmetry in project management, increasing the difficulty of coordination and supervision.

1. **Design Pattern**

In the face of the four challenges of complexity, conformity, changeability, and invisibility, the following are the design patterns related to the challenges of complexity and changeability:

**3.1. Component-Based Design Pattern**

The main purpose of the component-based design pattern is to reduce the complexity of systems and improve development efficiency by breaking down complex systems into multiple independent, reusable components. Each component encapsulates specific functionality and data, allowing developers to focus on the implementation of individual components without having to consider the details of the entire system. This pattern aims to promote modular design, thereby enhancing the maintainability and scalability of software.(1)

**3.1.1Advantages**

The advantages of the component-based design pattern include reduced complexity, increased reusability, ease of testing, and enhanced flexibility and scalability. By splitting the system into smaller, independent components, developers can more easily understand and manage the functionality and interactions of each component. Additionally, components can be reused across different projects, reducing the workload of redundant development, and each component can be tested independently to ensure its functionality is correct.

**3.1.2Practical Applications**

The component-based design pattern has been widely applied in enterprise applications, microservices architecture, and front-end development. In enterprise applications, different business functions are encapsulated as independent components, improving development efficiency and flexibility. In microservices architecture, microservices are treated as components that can be independently updated and extended, allowing the system to quickly adapt to changing business requirements. In modern front-end development, frameworks like React and Vue.js enable developers to create reusable UI components, thereby enhancing user experience and development efficiency.(2)

**3.2Strategy Pattern**

The Strategy Pattern is a behavioral design pattern that aims to achieve flexible behavior changes by encapsulating algorithms or behaviors within independent strategy classes. This pattern allows clients to select the desired algorithm at runtime without modifying the code that uses the algorithm, making it particularly suitable for scenarios where different algorithms or behaviors are needed, thereby effectively enhancing the system's changeability and extensibility.(3)

**3.2.1 Advantages**

The advantages of the Strategy Pattern include flexibility, separation of concerns, adherence to the Open/Closed Principle, and reduction of conditional statements. It allows for the dynamic selection of different algorithms at runtime, enabling the system to quickly adjust based on specific needs and enhancing developers' ability to respond to changing requirements. Additionally, by encapsulating different algorithms in independent strategy classes, the Strategy Pattern achieves clear separation of responsibilities, improving code readability and maintainability while reducing system coupling.(4)

**3.2.2 Practical Applications**

In practical applications, the Strategy Pattern is widely used in various fields, such as sorting algorithms, payment methods, graphics rendering, and data compression. In these scenarios, the Strategy Pattern enables users to select the appropriate strategy based on specific needs without modifying the core logic, thereby improving the system's flexibility and maintainability to meet the ever-changing business requirements.

1. **Case Study Analysis**

**4.2 Strategic patterns address variability challenges in energy management of commercial complexes**

**4.2.1 Case Background**

The large-scale commercial complex integrates office, entertainment, shopping and other functions, and the electricity consumption time of each region varies greatly, and is affected by the peak and valley electricity pricing policy. At the same time, electricity prices, business activities, equipment situation is constantly changing. In order to reduce costs and improve efficiency, the complex is equipped with energy storage equipment and adopts strategic mode to construct energy management system.

**4.2.2 Implementation of Strategy Pattern**

1. Defining the Policy interface

* The core method integrates energy usage, electricity price, energy storage equipment status and other related information (such as holidays, weather), and outputs the charging and discharging plan of energy storage equipment and the regulation scheme of electric equipment in each region, which provides a unified framework for the implementation of the strategy.

1. Implement concrete policy classes

* Arbitrage strategy of peak and valley electricity price: According to the change of peak and valley electricity price, charge and discharge in the valley and plan charging and discharging in combination with real-time demand and energy storage equipment status to reduce electricity cost.
* Time-sharing control strategy of electrical equipment in the office area: according to the working time and personnel activity law of the office area, the air conditioning and lights should be operated normally when working, and the unnecessary lights should be turned off when not working, so as to reduce the air conditioning power and save energy.
* Dynamic control strategy of electrical equipment in entertainment area: According to the business hours, passenger flow and electricity cost of entertainment area, the equipment is operated at full power during peak hours, and the power is reduced when passenger flow is low, balancing customer experience and energy utilization.

1. Creating a context class

* Set up an energy management context class that holds specific policy instances. By calling its execution method, the system can dynamically select and execute the corresponding policy according to the actual situation at runtime, and switch the energy management policy flexibly.

**4.2.3 Addressing the challenges of variability**

1. Electricity price policy adjustment: the peak and valley electricity price changes frequently, and the traditional way adjustment is complex. The strategy mode only needs to adjust the charging and discharging judgment conditions of the peak-valley electricity price arbitrage strategy, and the system can quickly adapt to the new electricity price policy.
2. Changes in commercial activities: The activities of the commercial complex are diverse and the electricity demand varies greatly. For example, when shopping promotion, the dynamic control strategy of electrical equipment in entertainment areas can adjust the power consumption scheme of equipment in real time according to the passenger flow and operation time, and easily respond to changes in energy demand without major changes in the system.
3. Equipment renewal: New equipment energy consumption and operating parameters are different. Taking the air conditioning update as an example, the control strategy of electric equipment in office and entertainment areas can adjust the operating power and control logic according to the characteristics of the new air conditioning, so that the system can adapt to the equipment update and ensure the energy management effect.

**Reference:**

1. .S. S. Yau and N. Dong, "Integration in component-based software development using design patterns" in Proceedings of the 24th Annual International Computer Software and Applications Conference (COMPSAC 2000), Los Alamitos, CA, USA: IEEE, 2000, pp. 369-374.
2. M. Babiuch and P. Foltynek, "Implementation of a Universal Framework Using Design Patterns for Application Development on Microcontrollers," Sensors, vol. 24, no. 10, pp. 3116-3143, May 2024.

(3)P. Hoverstadt and L. Loh, Patterns of Strategy. London, UK: Routledge, Taylor & Francis Group, 2017, 385 pp.

(4)E. Hewitt, Technology Strategy Patterns: Architecture as Strategy. Sebastopol, CA, USA: O'Reilly Media, Inc., 2018, 281 pp.