Рефакторинг программного кода

В современном мире существует большое множество языков программирования, которые позволяют создавать новые и усовершенствовать, ранее созданные, программы и системы.

Программный код — это текст, написанный на языке программирования. Разработчикам редко удается сразу написать хорошо структурированный код. Они выполняют задачу как можно быстрее, в процессе могут меняться требования к ней, тестировщики находят ошибки, которые нужно быстро исправить, или возникают срочные доработки, и их приходится делать очень быстро.

В результате, даже изначально написанный хороший код, становится непонятным и «грязным». Причем неважно, чужой это код или собственный.

Чтобы решить все эти проблемы, осуществляется рефакторинг программного кода.

Рефакторинг – это переработка исходного кода программы, чтобы он стал более простым и понятным. Рефакторинг не меняет поведение программы, не исправляет ошибки и не добавляет новую функциональность. Он делает код более понятным и удобочитаемым.

Рефакторинг выполняет при следующих проблемах:

– «мертвый код». Переменная, параметр, метод или класс больше не используются: требования к программе изменились, но код не почистили. Мёртвый код может встретиться и в сложной условной конструкции, где какая-то ветка никогда не исполняется из-за ошибки или изменения требований. Такие элементы или участки текста нужно очистить.

– дублирование. Один и тот же код выполняет одно и то же действие в нескольких местах программы. Здесь требуется вынести эту часть в отдельную функцию.

– имена переменных, функций или классов не передают их назначение. Имена должны сообщать, почему элемент кода существует, что он делает и как используется.

– слишком длинные функции и методы. Если получается больше 10-20 строк кода, требуется разделить функцию на несколько маленьких и добавить одну общую. Здесь маленькие функции выполняют по одной операции, а общая функция их вызывает.

– слишком длинные классы. То же самое. Оптимальная длина класса – 20–30 строк. Следует разбить длинный класс на несколько маленьких и включить их объекты в один общий класс.

– слишком длинный список параметров функции или метода. Если все эти параметры действительно нужны, нужно вынести их в отдельную структуру или класс с понятным именем, а в функцию передать ссылку на него.

– много комментариев. При желании пояснить какой-то участок кода, стоит попробовать сначала его переписать, чтобы и так стал понятным. Бесполезные комментарии загромождают программу.

Рефакторинг стоит выполнять постоянно и немного. Выполнять рефакторинг очень редко приведет к многочисленным ошибкам в системе, но в то же время, улучшать код постоянно.

Во время рефакторинга программного кода серверной части API «МОСП» были решены следующие проблемы:

– множественное комментирование;
– «мертвый код».

На рисунке 1 приведен модуль «MedCardController.js» до рефакторинга. Здесь проблема заключается в исчерпанном комментировании.

```
//  Создание новой мед карты
export const createMedCard = async (req, res) => {
    //  Обработка ошибок
    try {
        //  Получение значений с запроса
        const { idEmployee } = req.body

        //  Создание новой записи в документе
        const newMedCard = new MedicalCardModel({
            idEmployee
        })

        //  Сохранение изменений
        await newMedCard.save()

        //  Отображение добавленной мед.карты
        console.log('Send data from controller')
        return res.json(newMedCard)

    } catch (error) {
        console.log(error)
    }
}
```

Рисунок 1 – Модуль «MedCardController.js» (излишнее комментирование)

Рисунок 2 отражает решение данной проблемы. В данном случае приводится удаление подробных комментариев и добавление одного общего комментария, отвечающий за общую функцию.

```
//  Создание новой мед карты
export const createMedCard = async (req, res) => {
    try {

        const { idEmployee } = req.body

        const newMedCard = new MedicalCardModel({
            idEmployee
        })

        await newMedCard.save()

        console.log('Send data from controller')
        return res.json(newMedCard)

    } catch (error) {
        console.log(error)
    }
}
```

Рисунок 2 – Модуль «MedCardController.js» (общее комментирование)

Во ходе изменения функциональности и редактирования кода была решена проблема «мертвого кода». Рисунок 3 отражает код до внесений изменений. После редактирования нет необходимости получать объект «recordsInfo», который отвечает за записи мед.осмотра.

```
//  Создание новой мед карты
export const createMedCard = async (req, res) => {
    try {

        const { idEmployee, recordsInfo } = req.body
        const newMedCard = new MedicalCardModel({
            idEmployee,
        })
        await newMedCard.save()
        console.log('Send data from controller')
        return res.json(newMedCard)
    } catch (error) {
        console.log(error)
    }
}
```

Рисунок 3 – Модуль «MedCardController.js» (отражение проблемы «метрового кода»)

Необходимо удалить неиспользуемые объект для освобождения какой-либо памяти. Решение представлено на рисунке 4

```
//  Создание новой мед карты
export const createMedCard = async (req, res) => {
    try {

        const { idEmployee } = req.body
        const newMedCard = new MedicalCardModel({
            idEmployee,
        })
        await newMedCard.save()
        console.log('Send data from controller')
        return res.json(newMedCard)
    } catch (error) {
        console.log(error)
    }
}
```

Рисунок 4 – Модуль «MedCardController.js» (удаление лишнего объекта)