ШЕСТЬ ОШИБОК ПРИ ПРОВЕРКЕ ПРОГРАММНОГО КОДА

SIX MISTAKES IN SOFTWARE CODE VALIDATION

Авторы: *Бахтин Игорь Владиславович (САФУ имени М.В. Ломоносова)*

Аннотация: *В данной статье описываются возможные ошибки программистов при написании и анализе кода, а также возможные способы устранения этих самых ошибок.*

Ключевые слова: *Форматирование кода, рецензирование кода, рефакторинг, код-ревью, стиль программирования*

Annotation: *This article describes possible errors of programmers when writing and analyzing code, as well as possible ways to eliminate these very errors.*

Keywords: *Code formatting, code review, refactoring, revue code, programming style*

Проверка кода действительно может изменить качество программного продукта. Есть одно условие – программист должен сделать это правильно, чтобы реализовать свой потенциал. Рассмотрим наиболее распространенные ошибки, которые допускаются при ревью кода.

1. **Ориентируясь на стиль и синтаксис**

Этот тип обзора можно увидеть в большинстве проектов: много внимания на том, как что было написано, и никаких комментариев по более важным вопросам. Эта ошибка совершается часто, потому что эти дефекты действительно легко выявить. Глаз каждого разработчика используется для исправления дополнительных пробелов, ненужных линий и т.д. Это получается автоматически.

Как это должно работать:

Если программист делает что-то автоматически, машина сделает это еще лучше. Лучше всего утвердить одно руководство по стилю кода и использовать определенные инструменты для автоматического улучшения кода. Все разработчики в компании должны использовать его, независимо от того, работают ли они на Vim, Visual Studio или intelliJ IDEA. Инструменты для анализа кода могут обнаружить множество ошибок, и они делают это лучше, чем любой человек. Определение руководства по стилю кода и анализ его должны быть необходимым условием для хорошего обзора кода [1].

Обзор должен начинаться с синтаксических упрощений, не обнаруживаемых автоматическими инструментами. Легко сосредоточиться на незначительных ошибках и опечатках. Но главная цель состоит в том, чтобы понять общую картину и улучшить ее.

2. **Не вовремя**

Полчаса до демонстрации – худшее время для просмотра кода. Часто это происходит так: вместе с запросом на рецензию разработчик получает сообщение от заказчика, что необходимо внести правки в продукт. Правильный обзор требует времени и спокойствия, поспешный может не гарантировать качество. Такие ситуации часто возникают из-за плохого менеджмента, поэтому программисты не чувствуют ответственности. Независимо от того, кто виноват, какой-то плохой код был слит не в ту ветвь и, вероятно, останется там.

Как это должно работать:

У разработчика должна выработаться хорошая привычка: закрывать свои задачи задолго до презентации, выпуска или любого другого крайнего срока. Ничто другое не даст достаточно времени для обзора и, что более важно, для необходимых изменений.

Иногда происходит большая неудача, и представление новой функции – это вопрос жизни и смерти. В этом случае необходимо попробовать подготовить сборку из другой ветви, чем та, в которой проходит весь проверяемый код, вместо того чтобы ускорять проверку. Необходимо заранее обдумать эту возможность и соответствующим образом подготовить процессы построения и развертывания.

3. **«Не собираюсь заниматься дизайном»**

Еще одна ошибка, которая совершается разработчиками, не принимая всерьез в роли рецензентов самих себя. Легче сказать, что какой-то код будет работать правильно, чем судить, хорош ли его дизайн. Для этого необходимо тщательно продумать роль кода в системе и то, как он вписывается в существующую часть программы. Отслеживание совместимости с хорошими практиками, например, ООП, требует большого внимания. [2]

Как это должно работать:

С одной стороны, надо проверять, соответствуют ли новые компоненты или изменения высоким стандартам. Хорошо ли они спроектированы? Следующий шаг – переход на более высокий уровень – воздействие на архитектуру. Лучше всего проконсультироваться с другими членами команды.

4. **Неясные комментарии**

Комментировать строку, которая не нравится, с помощью “Fix please” бесполезно. Как кто-то может знать, что имел в виду рецензент? Программист, написавший данный фрагмент, может не понять, что именно не так. Может быть, он и поймет это, может быть, у них нет достаточных знаний, чтобы исправить фрагмент кода, может быть, он не понимают, что именно нужно исправить. Невозможно знать, что необходимо исправить, пока не будет конкретного замечания по коду. [3]

Как это должно работать:

Обзор кода – это также место для противостояния различным изменениям в коде.

Другое не всегда означает неправильное. Чтобы создать пространство для диалога, необходимо определить проблемы и предложить некоторые решения. Без них нет никаких оснований для обсуждения или изменения мнения. Кроме того, благодаря более широким описаниям обзор кода может стать местом, где программисты могут учиться друг у друга.

Обзор кода – это отличный инструмент, один из самых полезных. Необходимо использовать его, чтобы улучшить качество кода, противостоять идеям или передать свои знания. Требуется извлечь из этого максимум пользы для личного роста программиста.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Почему роботы должны форматировать код за нас – habr [Электронный ресурс].– Режим доступа: https://habr.com/ru/post/340304/ (дата обращения: 28.07.2020)

2 Оформление кода | Учимся писать чистый код – shwanoff [Электронный ресурс].– Режим доступа: https://shwanoff.ru/code-design/ (дата обращения: 28.07.2020)

3 Комментирование кода: хорошие, плохие и отвратительные комментарии – tproger [Электронный ресурс].– Режим доступа: https://tproger.ru/articles/comments-in-code/ (дата обращения: 28.07.2020)