**МИНОБРНАУКИ РОССИИ**

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Гжельский государственный университет»** (ГГУ)

Колледж ГГУ

Специальность 09.02.07. Информационные системы и программирование.

**ПРАКТИЧЕСКАЯ РАБОТА №6**

**по предмету: «Управление проектами»**

**на тему: «Ревьюирование программного кода code review»**

ВЫПОЛНИЛА:

Студентка группы ИСП-О-17

Пигарева Е.А

ПРОВЕРИЛА:

Прокуронова А. Ю.

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

п. Электроизолятор

2019 г.

**Стандарт код-ревью**

Основная цель код-ревью заключается в том, чтобы гарантировать постоянное улучшение кодовой базы Google. Все инструменты и процессы посвящены этой цели.  
Здесь необходим ряд компромиссов.

Во-первых, разработчики должны быть в состоянии успешно *решать свои задачи*. Если вы никогда не отправляете код, то и кодовая база никогда не улучшится. Кроме того, если рецензент сильно затрудняет *любую* работу, то в будущем разработчики не заинтересованы предлагать улучшения.  
С другой стороны, обязанность рецензента убедиться, что качество CL не снизит общее качество кодовой базы со временем. Это может быть сложно, потому что часто деградация происходит из-за небольшого снижения качества кода со временем, особенно если команда находится под сильным давлением сроков и чувствует, что имеет право на увеличение технического долга.

Кроме того, рецензент несёт ответственность за рецензируемый код. Он хочет убедиться, что кодовая база остаётся последовательной, поддерживаемой и соответствует всему остальному, что упомянуто в разделе [«Что проверять в коде»](https://habr.com/ru/post/467039/#2).

Таким образом, мы получаем следующее правило в качестве стандарта для код-ревью:  
**Обычно рецензенты должны одобрить CL, как только он достигает состояния, когда определённо улучшает общее качество кода системы, даже если CL не идеален.**

**Что такое качественный код**

Не существует точного определения этого термина. Как правило, понимание того, как должен выглядеть качественный исходный код, основывается на многолетнем опыте специалиста. Некоторые программисты придерживаются абстрактного принципа KISS, который расшифровывается как Keep It Simple, Stupid! («Делай это проще, тупица!»). Отчасти этот метод проектирования справедлив, так как отражает главное правило хорошего кода — простота и ясность. Однако простоту часто путают с упрощением, поэтому о качестве исходного кода в профессиональной среде судят ещё по нескольким свойствам:

* восприятие. Код не перегружен сложными конструкциями, поэтому его легко понять даже без дополнительной документации или комментариев;
* сопровождение. В продуманный код легко вносить изменения: менять конфигурации или даже платформы;
* расширение. В него просто добавить новую функциональность без риска сломать алгоритм кода. Даже если возникнут какие-то неполадки, их можно быстро устранить;
* передача. Хороший код можно передать другим разработчикам для поддержки или доработки, и у них не возникнет трудностей с его прочтением;
* покрытие тестами. Чем выше процент покрытия кода тестами, тем больше вероятность избежать ненужных багов в будущем.

Чтобы облегчить понимание кода в профессиональной среде, у каждого языка программирования есть свой Code Style — стандарт оформления. Именно он диктует правила: где ставить пробелы или скобки, как отделять строки или называть переменные. Может показаться, что эти нюансы не так важны, однако их соблюдение значительно облегчает понимание кода для тех, кто видит его впервые.

Написать по-настоящему хороший код может не каждый программист. Особенно тяжело это даётся тем, кто только набирается опыта. Но даже грамотные разработчики время от времени могут совершать ошибки. Поэтому студии, которые создают софт высокого качества, регулярно проводят инспекцию кода.

**Менторинг**

Код-ревью может быть важно ещё и для обучения разработчиков чему-то новому о языке, структуре или общих принципах проектирования ПО. Всегда приятно оставлять комментарии, которые помогают разработчику узнать что-то новое. Обмен знаниями вносит свой вклад в улучшение кода системы с течением времени. Просто имейте в виду, что если оставляете чисто образовательный комментарий, не критичный для соответствия описанным здесь стандартам, добавьте к нему префикс *Nit:* или иным образом укажите, что автор не обязан его разрешать.  
Принципы

* Технические факты и данные перевешивают мнения и личные предпочтения.
* В вопросах стиля абсолютным авторитетом является [руководство по стилю](http://google.github.io/styleguide/). Любая чисто стилевая деталь (пробел и др.), что не входит в руководство по стилю, является вопросом личных предпочтений. Стиль должен соответствовать тому, что есть. Если нет предыдущего стиля, примите авторский.
* **Аспекты программного дизайна практически никогда не проблема чисто стиля или личных предпочтений.** Они основаны на основополагающих принципах и должны определяться по этим принципам, а не просто на личном мнении. Иногда есть несколько допустимых вариантов. Если автор может продемонстрировать (либо с помощью данных, либо на основе твёрдых инженерных принципов), что определённые подходы одинаково эффективны, рецензент должен принять предпочтение автора. В противном случае выбор диктуется стандартными принципами разработки.
* Если никакое другое правило не применимо, то рецензент может попросить автора соблюдать единообразие с текущей кодовой базой, если это не ухудшает общее состояние системы.

**Как повысить качество кода?**

Одна из самых популярных и при этом довольно простых в реализации техник носит название Code Review. Её смысл в том, чтобы любые изменения, вносимые программистом, попадали в основное хранилище кода и в релизную версию ПО только после того, как их проверят остальные участники команды.

**Этот процесс состоит из нескольких этапов.**

Сначала разработчик добавляет новую функциональность в код и извещает остальных участников о том, что нужно проверить эти обновления.

На втором этапе члены команды, или ревьюеры, отсматривают код и оставляют свои комментарии. Некоторые компании, практикующие Code Review, фокусируются только на поиске багов, но для реального повышения качества кода нужно также указывать на архитектурные недочёты, неправильное использование инструментов и плохой стиль написания — непонятный или плохо воспринимаемый.

Дальше следует работа с замечаниями. Если автор не согласен с какой-топретензией, он может её отклонить, но для этого предстоит привести убедительные аргументы в защиту своей позиции. Если аргументов нет, он делает нужные исправления.

Дальше всё повторяется сначала и происходит систематически — каждый раз, когда в код вносится новая порция изменений.

**Плюсы Code Review**

Техника Code Review помогает на ранних стадиях находить некоторые ошибки и избавляться от непонятных и запутанных решений. В работе над кодом участвует не один человек, а целая команда, поэтому часто может появиться свежий взгляд со стороны.

Программист, который заранее знает, что коллеги проверят его работу, стремится писать более аккуратно и организованно. На выходе получается код, который понимают несколько человек, а значит, он намного ближе к качественному.

Когда группа из нескольких специалистов знакома с кодом на высоком уровне, его становится легко передавать между участниками процесса. Если возникнет необходимость, любой член команды сможет быстро углубиться в работу и сделать её качественно.

Благодаря Code Review снижается так называемый bus-фактор, или «фактор автобуса». Так называют число, означающее количество участников команды, которых должен сбить автобус, чтобы все знания о проекте были потеряны. К примеру, в проекте занято четыре человека, если два из них по каким-топричинам уйдут, то оставшиеся смогут закончить работу, а если команду покинут трое — последний участник не справится в одиночку.

**Минусы Code Review**

По сути, главный и единственный минус этого процесса — его длительность. Всем участникам Code Review приходится тратить время на то, чтобы посмотреть и при необходимости прокомментировать код, а разработчику — на исправление ошибок.

Новые дополнения попадают на этап тестирования не сразу, а только после прохождения review, из-за чего немного сдвигается график внутри этапа. Этот момент необходимо учитывать при планировании.

**Когда использовать Code Review?**

Процесс Code Review достаточно прост, а его плюсы заметно преобладают над минусами, но в ряде ситуаций вы легко обойдетесь без него. К примеру, нет смысла проводить Code Review при разработке прототипа или MVP — минимально жизнеспособного продукта. Главная задача такого проекта — получить от пользователей обратную связь, чтобы построить гипотезы для дальнейшего развития. Структура этих приложений делается максимально простой, и в дальнейшем код всё равно предстоит переписывать кардинальным образом. Ещё Code Review не нужен в работе над простыми приложениями, которые делаются раз и навсегда. Так что если вы не планируете в будущем изменять или дорабатывать свой проект, можно сэкономить время.