PowerIT, Mihai Valentin, 2022

Обзор на “код ревью”

Введение

Как показывает практика, почти все проекты на той или иной стадии разработки подвергаются изменениям. Это довольно очевидный вывод, ведь невозможно предугадать будущее. Ввиду бурного развития IT индустрии, умение адаптировать и изменять решения становиться критичным. Поэтому очень важно иметь налаженный процесс внедрения обновлений. Выверенные решения и шаблоны позволяют разработчикам легко понимать друг друга, передавать знания в команде и за ее пределами.

Одной из практик направленных на создание и поддержание таких условий является “код ревью” (англ. “code review”). Этот процесс подразумевает предварительный анализ и корректировку предложений по изменению кода, до их слияния с общей базой. Такой подход повышает стабильность, однородность и общее качество, а также снижает количество потенциальных дефектов в системе.

История код ревью в PowerIT

Для более глубокого понимания процесса проверки кода необходимо изучить натуральный путь развития этой практики. Для примера, предлагаю рассмотреть история код ревью в компании PowerIT.

Перед тем как перейти к анализу, хочу отметить, что для наглядности все процессы будут представлены в упрощенном варианте. Так например, не будет затронута интеграция отделов тестирования и контроля качества. Представим, что есть только разработчики, которые отправляют свои решения сразу в общую базу.

Никаких проверок

Для начала обозначим абстрактную точку отсчета - период, когда в принципе никакого контроля изменений не существует. Такая разработка неизбежно превращается в хаос, что приводит к конфликтам и провалам проектов. Естественно, на практике такая ситуация почти не встречается.

Из позитивного в данной ситуации можно отметить, что процесс разработки ничем не блокируется и протекает относительно быстро.

Самоконтроль

Самостоятельная проверка решений это безусловно лучше, чем ничего. Однако, и такой подход не лишен изъянов. Примерно так, когда-то давно, началась история код ревью в PowerIT.

Очевидные проблемы этого решения заключаются в необъективности оценки. В общем случае, ни один человек не сможет адекватно оценить свою работу. На такую оценку влияют как эмоции, так и “свежесть” взгляда. Чем больше времени разработчик проводит за решением и его проверкой, тем более рациональным оно ему кажется (как правило).

В список более сложных проблем входят: создание ненужного функционала, непредсказуемость, несогласованность и неоднородность общей базы.

Первая проблема заключается в том, что из-за отсутствия стороннего мнения, разработка может вестись над несуществующей задачей. Остальные же связаны с тем, что каждый разработчик (как правило) будет считать свои решения наиболее оптимальными.

Положительным является то, что такие “проверенные” решения потенциально менее рисковые, а с набором знаний о различных ошибках оценки становятся более объективными. Однако, этот подход не является лучшим.

Централизованная проверка готовых элементов

Проверка уже интегрированных решений не является продуктивным подходом. Как правило, такие проверки организуются по расписанию и за них отвечает специально назначенным специалист.

Именно этот человек и является слабым местом, так как всё упирается в его объективность, компетенцию и доступность. Его решения могут быть подвержены влиянию эмоций или каких-то личных взаимоотношений.

С другой стороны процесс проверки постфактум не сильно отличается от отсутствия контроля в принципе. Очевидно, что в адекватном процессе проверка предшествует слиянию.

Эти минусы можно скорректировать внедрением промежуточной среды тестирования и выбором действительно хорошего специалиста для этой задачи. Но к сожалению это не решает проблему продуктивности, непредсказуемости и необъективности.

В PowerIT такой сценарий проверки кода существовал совсем недолго и неизбежно привел к кризису, который и вызвал развитие этого процесса.

Сессии командных проверок

Следующей целью в развитии код ревью в PowerIT стало повышение объективности оценок. Такого результата можно достичь, например, за счет увеличения количества разработчиков, участвующих в процессе. Также нужно было изменить процесс так, чтобы проверки проводились до интеграции решений.

Таким образом, расширяя предыдущий подход, казалось бы, достигается и объективность, и предсказуемость. Однако, на практике это не так. В большой группе цензоров чаще возникают конфликты мнений, что может перерасти в личную неприязнь. Также, повышается и эмоциональное давление, ведь при всеобщем обзоре или страхе перед авторитетом рациональное решение может быть проигнорировано.

Такая зависимость от эмоционального состояния команды и каждого ее отдельного участника аннулирует как объективность, так и предсказуемость. Другим весомым недостатком является сложность в планировании таких масштабных мероприятий и их длительности.

В PowerIT такой процесс имел позитивное влияние в краткосрочной перспективе. Командные проверки и анализ кода позволил выделить и утвердить общие стандарты разработки и оценки качества.

Проверка со стороны старших разработчиков

После анализа кода большими группами был принят другой подход - проверка перешла под ответственность старших разработчиков.

Учитывая весь накопленный опыт, созданные стандарты и соглашения внутри команды, такой подход показал себя довольно хорошо. Он всё ещё сохранял зависимость от взаимоотношений в команде и эмоций, однако стал более объективным и предсказуемым (в общем случае).

Так как старшие разработчики являются двигателем команды, они как никто другой подходят для контроля качества кодовой базы проекта. Слабым местом здесь является численность таких разработчиков. В ситуации когда на каждого проверяющего приходиться двадцать проверяемых рабочая нагрузка становится невыносимой.

В PowerIT данный способ показывал довольно стабильные результаты, однако распределение нагрузки оставляло желать лучшего. Это и запустило новую волну изменений.

Проверка запросов на слияние

Анализ способов более равномерного распределения нагрузки привел к значительным изменениям в смежных процессах. Например, в таких, как декомпозиция задач и их приоритизация. В команде были созданы новые стандарты структурирования задач. По итогу, задачи стали более гранулированными, что позволило применить “ленивый” сценарий проверки новых изменений. Другими словами, пока одни задачи проверяются разработчик может перейти к следующим задачам и избежать блокировки процесса.

Далее развитие затронуло вопросов об удобстве, мониторинге и централизации всех проверок. Очевидным решением стало создание правила о том, что любые изменения должны внедряться через запросы на слияние.

В качестве хостинга проектов в PowerIT используется GitHub. Данная платформа предоставляет весьма удобный интерфейс для командной работы и проверки кода, в том числе. Так, разработчики могут выбирать проверяющего. Проверяющий же может оставлять комментарии прикрепленные к конкретным строкам кода. Такой простой механизм обсуждения решений делает процесс максимально прозрачным.

Эти шаги позволили сделать проверку более удобной и менее трудозатратной. Также, появилась некая “история” хода проверок и стандартизированный подход к запросам на слияние. Как бонус, снизилось влияние эмоций и личностных отношений, так как все проверки стали проходить в форме текстового общения.

Другим важным фактом является то, что этот процесс удобно интегрируется с процессами CI/CD и легко расширяется за счет автоматизации.

Автоматизированная проверка

Автоматизация рутинных процессов стала следующим этапом эволюции. Учитывая что в команде уже существовал набор стандартов и “лучших” практик, не составило труда автоматизировать проверку соответствия этим стандартам.

Основой автоматизированной проверки стали операции по проверке сборки приложений (build test), контролю общего стиля написания кода, проверке соответствия внутренним стандартам, анализу потенциальных ошибок (с помощью строгого статического анализа кода), проверке наличия всех необходимых компонентов (например, наличие документации) и автоматическое тестирования кода (unit и feature тесты).

Естественно такой вариант проверки не может считаться достаточным, ведь он не покрывает сложные или нестандартные задачи. Также, он не позволяет принимать решения с учетом внешних факторов (например, пожар). С другой стороны, такой процесс требует минимум времени (при должной оптимизации).

После того как авто-проверки были приведены к стабильному и оптимальному (по времени выполнения) состоянию, было принято решение создать комбинированный подход, который включал бы все плюсы предыдущих и не содержал бы очевидных минусов.

Комбинированный подход

В составлении комбинированного подхода основными целями стали: объективность, децентрализация, комфорт, скорость и предсказуемость.

Объединив командные проверки, проверки старшими разработчиками и автоматизированные проверки был получен необходимый нам процесс контроля качества кода.

Из первого была взята модель принятия решений голосованием, например: для прохождения проверки необходимо два одобрения. Вес голосов и их количество настраивается в зависимости от сложности и потребности в изменениях. Однако, у каждого разработчика есть право на голос и такое же право на запрос оценки.

Голоса старших разработчиков, очевидно, имеют бо′льший вес и являются решающими в критических ситуациях. Это было унаследовано из второго подхода.

И наконец, автоматизация базовых проверок стала отличным дополнением в ускорении процесса.

Результат этого синтеза стал нормой проверки кода в PowerIT на текущий момент. Основной его минус заключается в сложности имплементации, так как требует работы не только над техническими аспектами, но и работы над отношениями в команде: уважение, доверие, взаимопонимание и взаимопомощь.

Результаты внедрения проверок

По мере продвижения по пути улучшения процесса проверки кода было получено множество полезных результатов:

- однородность и высокое качество кода;

- рациональное структурирование задач;
- документированные стандарты;
- автоматизация и ускорение процессов;
- сплоченная команда;
- минимум конфликтов;
- постоянный обмен опытом в команде;
- легкое внедрение новых разработчиков в процессы;
- автоматическая статистика;
- предсказуемость;
- портативность;
- открытость к расширениям.

Возможности расширения подхода

Так как любой процесс можно улучшать до бесконечности, следует отметить пути к расширению.

Первое, что приходит на ум при анализе описанных вариантов - это ограничение прав доступа на внесение изменений и строгий контроль показателей покрытия кода тестами, документацией и тд.

Другой веткой развития может быть автоматизация и улучшения интеграции с QA инженерами. Также существуют несложные способы интеграции различных проверок с локальной средой разработки (речь о IDE и других схожих инструментах).

И наконец, можно организовать различные рейтинги для создания эффекта “игры” в процессе разработки, однако подобные решения следует внедрять с осторожностью.

Заключение

В заключение хотелось бы отметить, что подобного рода проверки безусловно необходимы, но не стоит забывать, что доверие является первичным. Строгость процессов, особенно по контролю, будет обратно пропорционально влиять на отношения в команде.

Также, не стоит забывать, что “идеального” кода не существует. Поэтому нужно помнить об уважении, доверии и сплоченности в команде. Такой коллектив безусловно будет реализовывать лучшие решения, а их проверка будет протекать легче и быстрее.

Во всем важен баланс и адекватный, взвешенный подход.