Проблемы с кодом? Помогите команде писать лучший код

Порой мне кажется, что ближе к вечеру многие разработчики начинают думать, что некачественный код, особенно в крупном проекте, не такая уж большая проблема. И собственно, если вы решите сосчитать, сколько проектов провалилось из-за проблем с кодом, то насчитаете их вы, и правда, не так уж много. Так или иначе, думаю, вы не захотите ждать настоящей катастрофы, в результате которой вы потеряете уйму денег.
Будучи ведущим разработчиком (архитектором решений или, как вы там называетесь на визитке), что вы можете сделать для того, чтобы помочь команде писать лучший код?

Семь добродетелей разработки программного обеспечения
Осмелюсь сказать, что успешные проекты зависят от двух факторов: от руководства, знающего толк в лидерстве, и разработчиков, понимающих, что такое качественный код. Вот список семи добродетелей, которые по-настоящему могут привести команду в рай разработчиков.

№1 Пусть инструменты помогают в написании кода

Программирование не должно разбиваться на время, когда разработчики просто пишут код, и на время (позже), когда они этот код чистят и исправляют. Это «позже» не наступает никогда. Писать хороший код нужно сразу. И тут на помощь приходят инструменты. Обычно встраиваемые в среду разработки, эти инструменты упрощают типовые задачи, позволяя разработчикам работать быстрее и писать действительно лучший код. На худой конец, код пишется быстрее, за счет чего остается время на его исправление.
Автозаполнение, подсказки по идиоматичности кода (т.е. соответствию кода языку или фреймворку), инспектирование кода, заранее заданные фрагменты кода, вставляемые по нажатию сочетаний клавиш, и предопределенные и настраиваемые шаблоны – все это ускоряет разработку, обеспечивает согласованность и в какой-то мере лучший и более чистый код.
Ассистенты ввода кода делают разработку более эффективной и значительно улучшают качество вашего кода всего в несколько кликов. Они выявляют дублирующийся или неиспользуемый код, облегчают рефакторинг, упрощают навигацию и инспектирование и предлагают паттерны. Более того, инструменты помогают принять соответствующее соглашение об именованиях, делая переименование или рефакторинг метода пустячной задачей.
ReSharper – самый популярный из существующих ассистентов ввода кода. Дополнительную информацию можно почитать тут: *http://www.jetbrains.com/resharper*. Другие аналогичные инструменты – CodeRush от DevExpress (*http://www.devexpress.com/Products/CodeRush*) и JustCode от Telerik (*http://www.telerik.com/products/justcode.aspx*).

№2 Если кто-то пишет плохой код, скажите ему об этом

Предположим, вы заметили, что некоторые люди в вашей команде пишут плохой код. Как им об этом сказать?

Здесь нужно учитывать некоторые психологические аспекты. Не нужно быть резким и обижать кого-то. В то же время, вам вряд ли хочется отвечать за чью-то плохую работу. Лучший способ поговорить о коде, который вам не нравится, осторожно спросить человека, почему он написал его именно так. Это позволит вам понять его мотивацию и не было ли какой дезинформации, плохого отношения, неопытности или ограничений, о которых вы не догадывались.
Очень важно, чтобы вы не критиковали код, не имея для этого четких причин. Поэтому просто притворитесь, что вам любопытно, почему код был написан именно так, а интересуетесь вы потому, что сами бы написали его по-другому.

№3 Помогите разработчику стать лучше

Золотое правило, которому нужно следовать, чтобы команда выдавала лучший код, звучит следующим образом:

Критикуйте код, а не кодировщика, но улучшайте код через кодировщика.

Любой фрагмент плохого кода можно исправить. Если это требуется сделать, не стоит винить кодировщика, лучше помочь ему исправиться самому. Сделав это, вы получаете два преимущества. Вы получаете лучшего разработчика и, возможно, более счастливого и более мотивированного работника. Вы позволяете ему почувствовать себя чуть ли не героем, потому что ему удалось выполнить свою работу лучшим из существующих путей.

№4 Инспектируйте код перед подтверждением изменений

В вашей компании могут быть лучшие в мире стандарты кодирования, но как обеспечить их соблюдение? Доверять разработчикам, конечно, нужно, но нельзя забывать и проверять их работу. Парное программирование и систематический критический разбор решений являются конкретными способами проверки состояния базы исходных текстов. Для разбора решений можно взять некий образец кода и обсудить его всей командой. Это может быть реальный фрагмент кода из проекта, написанный одним из членов команды, или же – чтобы не задеть ничьи чувства – специально написанный кусочек кода, который демонстрирует то, что вы хотели сказать.
Чтобы обеспечить соблюдение стандартов кодирования, можно принудительно указать политику подтверждений в системе контроля исходного кода, будь то TFS, TeamCity и т.д. Можно ли как-то автоматизировать этот процесс?
Сегодня почти любой инструмент контроля исходного кода предлагает контроль файлов, передаваемых на подтверждение. Например, в TFS имеется система gated check-in – подтверждение изменений, осуществляемое по правилам.
Другими словами, файлы, проходящие подтверждение, принимаются в систему только если они отвечают установленным требованиям. При создании gated check-in TFS предлагает указать существующий скрипт билда, который будет использоваться. Только если билд завершается успешно, файлы подтверждаются. В TFS билд – это просто скрипт MSBuild, который может быть настроен на множество различных задач. TFS имеет большое количество предопределенных интегрируемых задач. Например, задачи анализа кода (в

прошлом FxCop) и задача запуска списка выбранных тестов. Поскольку MSBuild – не более, чем зарегистрированный компонент, который реализует контрактный интерфейс, вы можете создавать новые задачи сами, чтобы добавить свои правила валидации.
Стоит отметить, что вышеупомянутый ассистент ввода кода ReSharper от JetBrains в своей последней версии предлагает набор бесплатных инструментов командной строки, которые можно использовать в персонализированной задаче MSBuild для выявления дублированного кода и проведения типового инспектирования, включая персонализированное инспектирование по определенным шаблонам. Для того, чтобы использовать инструменты командной строки, вам не понадобиться лицензия ReSharper. Более подробную информацию по инструментам командной строки от ReSharper можно получить тут: *http://www.jetbrains.com/resharper/features/command-line.html*.

№5 Счастлив тот проект, которому не нужны герои

Разработчикам свойственно преувеличивать свои способности. По крайней мере, многие из нас хоть раз втайне представляли, как они работают более 80 часов в неделю, спасая проект, дарят радость заказчикам и выглядят истинными героями в глазах руководства и коллег. Бывает, что сроки определяются неправильно с самого начала. Иногда это понимание приходит лишь в середине проекта. В таких ситуациях всегда проще безропотно согласиться с таким положением вещей, но особого облегчения это не приносит и, что более важно, требует проявлений геройства. Суть прозрачности заключается в том, чтобы говорить о проблемах открыто, но в то же время – это и эффективный способ взять ситуацию под контроль и снизить давление.
В разработке ПО мы испытываем давление из-за поджимающих сроков или из-за отсутствия необходимых навыков. И та и другая проблема вполне решаемы, если вовремя сообщить о них. Не нужно доводить до того, чтобы потребовалось вмешательство героев. Я был таким героем несколько раз и урок, который я усвоил, заключается в том, что геройство – это исключительная ситуация, а в разработке программного обеспечения нужно стараться избегать любых исключительных ситуаций.

№6 Поощряйте тренировку

Чего ради профессиональные спортсмены тренируются каждый день по многу часов? И можно ли провести аналогию между разработчиками и профессиональными игроками? Когда как.
С одной стороны, разработчики и так тренируются каждый день на работе и не соревнуются друг с другом. А значит, никакой аналогии нет и, следовательно, тренироваться не надо.
С другой стороны, спортсмены отрабатывают базовые движения так часто, что могут повторить их на автомате. Постоянный возврат с основам объектной ориентации, шаблонам проектирования, стратегиям кодирования и определенным областям API помогает хранить знания наготове и использовать их максимально быстро.

№7 Непрерывное изменение – это часть сделки

Проект разработки программного обеспечения начинается с некой расплывчатой бизнес-идеи. Многие проекты напоминают движущиеся мишени, которые приводятся в движение требованиями. Непрерывные изменения – это привычные ежедневные новости, а не нечто принципиально новое. Очевидно, склад ума – единственный эффективный способ, помогающий справиться с этим потоком изменений. Чтобы быть эффективным, каждый разработчик должен быть готов к рефакторингу.
Рефакторинг – одна из тех вещей, которые, кажутся, не привносят в проект никакой ценности. К несчастью, без рефакторинга не будет и ценности.

Плохой код обходится дороже хорошего

В контексте продолжительного проекта, имеющего большое значение для бизнеса, плохой код обходится в разы дороже хорошего кода. Как бы просто это ни звучало, проект умирает из-за плохого кода, когда стоимость работы с ним (его создание, тестирование и сопровождение) превышает стоимость расходов, с которыми может справится модель предприятия. Справедливо и то, что никакие проекты не проваливаются из-за проблем с кодом, если компаниям удается удерживать стоимость расходов, связанных с кодом, на предельно низком уровне.

В этом-то и заключается главная проблема.

Иногда руководство сводит это к простому сокращению расходов на разработку, найму низкооплачиваемых разработчиков и к просьбам/распоряжениям сократить объем тестов и документации. Они снижают стоимость написания кода, превращая проект в домашнее задание. К сожалению, создание *кода-который-просто-работает* – лишь одна сторона проблемы. Сегодня требования меняются постоянно, увеличивается сложность – и, что хуже всег, по-настоящему осознается лишь на ходу. При таком сценарии написание кода – лишь одна статья затрат. Сопровождение кода и его развитие – наиболее затратные статьи. Хороший архитектор прекрасно знает, что сопровождаемость кода напрямую зависит от его качества, понимания принципов разработки ПО и особенностей языка, правильно применяемых паттернов и практик, и тестируемости. Это делает кодирование куда более дорогим процессом, чем создание *кода-который-просто-работает*, но намного дешевле сопровождения и развития *кода-который-просто-работает.*