Вступление

Что побудило меня выйти к вам сейчас и поговорить о плохом и хорошем коде? За время работы верстальщиком я работал над разными проектами, и большими, и маленькими, и очень большими. Обычно это был уже готовый продукт, который находился в стадии расширения, соответственно, основной код, архитектура, уже была, и нужно было в рамках уже существующего либо дописывать новый функционал, либо что-то менять.

Проекты, как я уже сказал, были совершенно разными по всем параметрам, и объединяло их, пожалуй, только одно – довольно несвежий, местами нелогичный и плохо поддающийся правкам код. Довольно часто возникали проблемы с тем, что внесенные изменения в одной части сайта ломали верстку в другой части сайта, связи были настолько неочевидными, и в то же время, они все-таки были, что порой было легче написать необходимый код с нуля и подключить его в нужном месте. Текущие проблемы это решало, но логичности архитектуре проекта не добавляло.

Так вот, давайте попробуем для начала разобраться, из-за чего код превращается из хорошего в плохой. Причины каждый из вас знает, не будем делать из этого открытие. Самая очевидная причина – дедлайны. Перед вами есть задача сделать некий функционал для проекта к определенному времени. И тут внезапно к вам подходит ПМ и говорит: «Тут такое дело, заказчик слезно просил запилить фичу на неделю раньше первоначального срока. Обещал премию!». Ты думаешь – здорово, и отвечаешь: «Да, конечно, только есть один нюанс. Из-за того, что сроки получаются ужатыми, у меня не хватит времени на то, чтобы интегрировать код нового функционала, придется сейчас писать, особо не оглядываясь на стандарты и логику остального проекта, чтобы запустить фичу в срок». ПМ соглашается с тобой: «Да, я понимаю, но мы не можем ему отказать. Давай мы все же сейчас сделаем эту фичу рабочей, чтобы не опоздать с релизом, а потом мы с тобой сядем и обговорим вопрос интеграции и рефакторинга». После этого разговора ты садишься и делаешь новую фичу, не особо оглядываясь на структуру и логику остального проекта, ведь сейчас самое главное – зарелизить фичу. Ты уложился в срок, все довольны и счастливы. На следующий день ты подходишь к Пму и напоминаешь о вашем разговоре – «Послушай, было бы здорово закончить то, о чем мы с тобой говорили. Мне на это нужно примерно три дня», но в ответ слышишь: «Знаешь, тут такое дело. Заказчик был в совторге от новой фичи, и на радостях дал новую порцию фич. Он пообещал также выделить премии, если все будет сделано в срок. Нам нужно уложиться и сделать то, что он просит. А вот после... мы обязательно вернемся к твоему вопросу». После этих слов ты недоумевающе смотришь на Пма, но делать нечего – нужно внедрять очередную фичу. Ты смотришь ТЗ, и оказывается, что при данных сроках ты не то, что не сможешь подобраться к интеграции своего кода в проект, - ты даже толком в этом новом коде не сможешь поправить логику, которая изменилась с учетом правок и новых условий в ТЗ. Времени есть только на то, чтобы писать код, чтобы успеть к релизу. Так повторяется еще несколько раз, ты уже даже не напоминаешь ПМу о том, что было бы здорово все-таки что-то сделать в направлении рефакторинга. Теперь ты после прочтения ТЗ о новых фичах сразу ходишь на перекур и выкуриваешь там по три сигареты, твоя мотивация стремится к нулю, и в итоге ты уходишь с проекта. И ведь вроде бы все были правы – был прав заказчик, - он же по дефолту прав, ведь он исправно платил за работу, выдавал обещанные премии. Был прав ПМ – ведь он называл заказчику реальные сроки, в которые его команда успевала разработать тот или иной функционал, который был нужен заказчику. И ты вроде был прав – ведь ты так же в срок успевал сделать свою часть работы над продуктом, оттого релизы не затягивались и всё сдавалось в срок. А проблема в этом всем была только одна – заказчика, в большинстве своем, интересует в первую очередь та сторона продукта, которая позволяет этому продукту быть востребованным на рынке, выполнять свои задачи и приносить деньги. Ведь он нанял вас не просто так, он доверяет вам как специалисту и уверен в том, что с вашей стороны в продукте тоже все здорово – и он вроде как не видит ничего, что заставляло бы его думать по-другому, ведь фици релизятся, работа идет, и все проссходит в срок, без задержек, следовательно, проблем в проекте нет. К сожалению, если поставить перед заказчиком или Пмом проблему несовместимости отдельных модулей в проекте, отсутствие связанной логики в процессах, множество мелких и не только костылей, которые были сделаны на разных этапах разработки продукта с тем, чтобы уложиться в срок, то скорее всего эта проблема будет отодвинута на задний план как наименее приоритетная. Ведь код писался раньше, фичи добавлялись? С чего это вдруг в один прекрасный день делать это стало почему-то значительно тяжелее. Ну, может быть, но если проблема обозначилась только сейчас, то, скорее всего, они может еще подождать своего решения, а мы пока зарелизим еще пару фич.

Также проблемы могут возникать из-за отсутствия нормальной документации к коду проекта. Новый человек, который только пришел в команду, конечно не знает всех логических связей, условностей, договоренностей, в конце концов, уже существующих костылей, и потому может начать писать свои просто потому, что не знает, как правильнее сделать то, что ему нужно сделать в рамках поставленной ему задачи. Хорошо, если есть человек, к которому можно обратиться, ведь часто бывает и наоборот – уже не осталось никого из тех людей, кто разрабатывал изначальный продукт, и ответы на свои вопросы можно искать разве что в истории коммитов.

// Пример

Когда я пришел в бигмир, пришлось звонить Наташе, чтобы узнать, как именно заводить конкурс на разделе фотографий

// End of

Ну и самая главная причина возникновения таких проблем – лень. Вы видите перед собой проблему с логикой проекта, и знаете, какие конкретно шаги нужно сделать, чтобы проблему решить, но вы понимаете, что на это может уйти, скажем, несколько часов, да и задача не слишком интересная, и ее вполне можно отложить «на потом». Вы даже отметили ее в трекере, чтобы не забыть и когда-нибудь к ней вернуться, но это время так никогда и не настанет. А потом внезапно станет поздно.

Какие правила позволяют избежать такой печальной участи?

Конечно, у проекта должна быть документация, в которой будет описан некий свод правил и договоренностей относительно написания кода, логики, структуры, архитектуры и остальных деталей проекта. Однако зачастую одной документации бывает недостаточно, потому что всегда есть вероятность возникновения тех проблем, о которых мы недавно говорили.

Опыт, документация проекта и «хороший код».

Зачем же нам нужна чистота кода и каких правил следует придерживаться, чтобы код оставался понятным и легким?

Код уже давно пишется не для машин, не для браузеров. Браузеру уже давно всё равно, каким образом вы сверстали тот попап, или запилили диаграмму, или построили сетку. Быстродействие браузеров уже давно вышло на такой уровень, что даже два совершенно разных кода, хороший и плохой, между которыми есть пропасть, практически одинаково рендерятся в браузере и в итоге получается одинаковый результат, даже незаметный рядовому пользователю.

Код уже давно стОит писать в первую очередь для людей – для разработчиков, которым приходится иметь дело с этим кодом, сейчас или в дальнейшем. Для себя, в конце концов. И чем лучше, человечнее, будет выглядеть этот код, тем меньше усилий потратит тот человек, который спустя год зайдет в этот код, найдет нужное ему правило, изменит его так, как нужно, и будет доволен. Вполне возможно, что этим человеком будете вы сами.

HTML

Давайте перейдем непосредственно к тем мелочам в коде, которые никогда не бывают лишними, но о которых часто забывают. И начнем с разметки, с HTML.

Стандарт HTML5 у всех на слуху уже лет 5. Все знают о короткой записи доктайпа, но немногие вспоминают о чем-то, кроме доктайпа. Стандарт HTML5 позволил нам, разработчикам, писать код совершенно свободно, без тех строгостей, которые были в стандарте HTML 4.01 и XML – атрибуты тегов теперь можно заключать в кавычки, а можно ине заключать, логические атрибуты теперь тоже можно записывать по-разному, одинарные теги можно закрывать в соответствии со стандартом XML, а можно и не закрывать. На мой взгляд, эта свобода, которая поначалу воспринималась всеми на уровне «вау», несет в себе больше проблем, нежели чего-то хорошего. Ведь теперь разные разработчики могут писать один и тот же код совершенно по-разному. Другое дело, что для браузера этот код будет одинаково правильным, но для конкретного разработчика это будет неким камнем преткновения.

Для лучшего восприятия кода и для его однообразия вам в команде следует оговорить все спорные моменты, которые разными разработчиками могут интерпретироваться по-разному. В идеале, если между двумя разработчиками существует некое разногласие, всей команде стоит собраться, чтобы обсудить плюс и минусы каждого из подходов, каждого и способов, и определить, какой именно способ стоит использовать. Это решение не всегда может опираться исключительно на доводы; - эти доводы могут быть взаимозаменяющимися и одинаково правильными, - в этом случае следует просто остановиться на чем-то одном.

Отдельно хочется упомянуть о новых HTML5 тегах. Некоторые до сих пор спрашивают «зачем они», другие просто относятся к ним с недоверием, третьи используются и везде и всюду, где надо и где не надо. Зачем это? Сейчас основной целью информации в интернете выступает ее доступность. Доступность на различных устройствах, программах, внебраузерной сфере. На что опираться таким устройствам и программам? Уж точно не на ваш, без сомнения, классный CSS, который оформляет весь код. Страница на таких устройствах должна выглядеть примелемо (читаемо) и без стилей. Такое утверждение известно еще и со времен HTML4, но сейчас, во-первых, в игру вошли поисковики. Новые теги позволяют вишему сайту быть более доступными для поисковиков и лучше и качественнее читать и воспринимать контент страниц. Плюс к этому, технологии не тостя на месте, семимильными шагами веб развивается в области доступности для людей с ограниченными возмоностями, - и скорее всего, именно разметка, включая и новые теги, будет главным пунктом качества доступности информации.

Обращайте внимание на заголовки, списки, кусочки текста, несущие основную смысловую нагрузку – всё это важно.

Вся информация о новых тегах и об их употреблении уже давно стала адекватной, очевидной и прозрачной. Читайте спеку, и да прибудет с вами сила.

Также не забывайте о такой мелочи как микроданные и микроразметка. Это также позволит сделать ваш сайт более доступным, а поисковикам – лучше обработать ваш сайт и правильно отобразить его в выдаче.

// Пример

Например, прошлый летом перед нами стояла довольно большая задача – нужно было разметить полностью весь раздел видео

// End of

CSS

В наше время к CSS уже давно пора относиться, как к любому из языков программирования, ведь он стремительно развивается и на данный момент уже обладает некими возможностями, напрямую позволяющими сравнить его с ЯП.

Другое дело, что верстальщик, человек, который пишет CSS, не является крутым только потому, что страница в браузере выглядит точно так же, как на макете. Ведь так должно быть, так написано в спецификации, и тот, что сайт сверстан в соответствии со специкификацией, не делает верстальщика крутым. Крутым верстальщика делает другое – качество кода. Качество – довольно субъективная вещь, однако ее можно оценить, сравнив два варианта одного и того же результата, проанализировав их и поняв, какой из вариантов обладает большей гибкостью, читаемостью, расширяемостью.

Эти вопросы вполне естественны при оценке кода, и CSS здесь не должен быть исключением. Сегодняшние веб-приложения становятся всё более крупными, поэтому слабая и прямолинейная архитектура CSS может сильно усложнить разработку. Пришло время отнестись к CSS так же, как мы относимся ко всем другим частям наших приложений. Это не та вещь, которая может быть оставлена на потом или сброшена со счетов как «дизайнерская проблема».

У сообщества нет единых соглашений касательно вопроса о том, как же все таки должен выглядеть хороший код, потому вместо какие-то глобальных правил стоит говорить для начала о принципах. Ведь если договориться о принципах, то возможно мы станем различать плохой CSS не только потому, что он противоречит каким-то нашим убеждениям, а потому что он действительно усложняет процесс разработки.

Чего стоит ожидать от хорошего CSS? Это, пожалуй, несколько основных моментов: предсказуемость, переиспользуемость и масштабируемость.

Предсказуемость означает, что правила ведут себя ожидаемо. Когда вы изменяете какое-то правило, оно не должно никак влиять на части сайта, которые вы не собирались менять.

Повторное использование означает, что CSS-правила должны быть достаточно абстрактными и независимыми, чтобы при необходимости создать новые компоненты из существующих частей не нужно было что-то переписывать.

Масштабируемость означает, что проект при своем росте можно будет поддерживать так же легко, как одним человеком, так и командой.

Самое плохое в CSS, что может с вами случиться – это то, что вы до сих пор пишете чистый CSS. Тут недавно Sass отпразновал свое вольмилетие, а вы все пишете CSS…

На данный момент существует несколько CSS-препроцессоров, каждый из которых довольно активно развивается, у каждого из них есть довольно серьезное сообщество. И если вы еще не сделали выбор, то сейчас самое время. CSS-препроцессоры предлагают широкий фуинкционал из того арсенала, который имеется в CSS, но, к сожалению, только в роли спецификаций, а не в части поддержки браузерами. CSS-препроцессоры временно заменили CSS, хотя сейчас, кажется они уже не отпустят свое. Среди их преимуществ – переменные, функции, миксины, циклы, и все это прямо из коробки. Проблема препроцессоров только в одном – с их помощью можно быстрее написать любой код, в том числе и плохой, который будет использовать как плюс лишь самые очевидные части препроцессора, как, например, каскад.

Плюсы препроцессоров очевидны, но в некоторых случаях эти плюсы также могут вылиться для проекта в минус, - например благодаря возможности несколькими спсобами унаследовать некоторые заренее определенные CSS-свойства. Ситуация получается очень похожей на случай со свободной разметкой в HTML5 нотации. В этом случае внутри команды аткже стоит определиться с тем, как именно могут наследоваться свойства, и использовать выбранный вариант в дальнейшем внутри проекта.

// Пример

Несколько вариантов наследования свойств

// End of

// Нужен пример со страшным наследованием, порядком свойств

Пользуйтесь средствами, улучшающими навигацию в коде. Помимо обычных комментариев, используйте source maps - сейчас вроде бы каждый препроцессор обладает их поддержкой. Да, у них есть минус - они несовместимы с минификацией кода. Здесь есть пара уловок - можно создавать оба варианта CSS и при необходимости подменять их, либо генерировать два разных файла – один сжатый, а другой несжатый, который отдавать при необходимости.

// Пример

В бигмире мы внедрили поддержку source maps и это работает сейчас на любой странице портала, где есть новый код, написанный на LESS. Ситуации, когда он необходим, случались очень редко, поскольку у нас попросту очень прозрачная и простая архитектура LESS, несмотря на большое количество файлов. Почему так сложно – потому что при тестировании оказалось, что, несмотря на то, что в то время в Хроме поддержка source maps была спрятана за флагом, .map-файл все равно грузился у абсолютно всех пользователей. Потому было решено подгружать всю эту логику только по требованию.

?sourcemaps=1

Также, поскольку в бигмире мы использовали LESS, я внедрил такую for fun штуку – возможность генерации CSS прямо в браузере – то есть клиенту отдается не скомпилированный CSS, а исходный LESS плюс подключается файл библиотеки, и генерация происходит прямо в браузере. Правда работает это только из офиса, и какого-то серьезного смысла пока не несет, у меня были планы часть стилей перенести на компиляцию прямо в браузере, но у меня не дошли руки до того, чтобы серьезно сесть и протестировать плюсы и минусы этого подхода на каком-нибудь кусочке кода.

?debug=less

// End of

Форматирование кода. Я попробую на одном примере показать всю ту боль, с которой мне приходилось сталкиваться.