Тезисы

Что такое «правила хорошего тона», в чем бы то ни было? Это свод неких устоев и правил поведения людей в каком-либо обществе.

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.

Давайте попробуем разобраться, почему практически в каждом проекте возникают некие проблемы, которые в будущем отражаются на удобстве и сроках разработки.

1. HTML

Использование html5 – это не просто короткий doctype, а целая куча других фишек. Глупо не пользоваться этими фишками.

Свободный кодстайл, который предоставляет новая версия HTML, хорош и удобен, но эта свобода таит в себе и угрозу – разные разработчики могут писать то, что раньше было строго стандартизировано, по-разному, натыкаясь на несоответствие в коде в самых неожиданных местах. Во избежание этого стоит остановиться на конкретном наборе правил и следовать ему.

1. Все имена тегов, атрибутов пишутся только строчными буквами.
2. Все значения атрибутов заключаются в скобки
3. Логические атрибуты записываются только как название атрибута без необходимости повторения того же самого в значении, либо без использования логических операторов
4. Не нужно писать атрибуты для script/style
5. Короткая запись мета-тегов
6. Одинарные теги закрываются согласно правилам XML – с использованием слеша (?) – сомнительная вещь, вряд ли о таком стоит говорить и советовать.

Помимо этого, можно договориться о многих других мелочах, настройку которых предоставляет, например, ваша IDE.

Вернемся к HTML5. Первая известная вещь после доктайпа – это появившиеся новые html-теги, призванные качественно улучшить описание структуры и контента документа.

Появились новые теги – но по-прежнему используются одни дивы. Отчасти из-за непонимания или неоднозначности формулировок, или правил в самой спецификации. Но структура и спецификация становятся все более прозрачными, в сети имеется большое количество примеров использования тех или иных тегов.

Но семантики в целом как не было, так и нет. Семантика – это не только попытка угодить поисковых системам, но и браузерам в целом.

Необходимо переосмыслить строение страницы и сделать его более семантичным, стараясь использовать теги, которые предназначены для отображения тех или иных элементов - <mark>, <ul>, <header>, <menu>.

Отдельно – использование различных фреймворков – особенно это касается бутстрапов/foundation. Проблема в недостаточном знании инструмента, его плюсов, минусов и границ адекватной работы. Попытка поменять что-то в поведении выливается в другие проблемы. Чем больше разрастается проект, тем больше костылей на нем становится.

В чем проблемы появления костылей? В том, что новый человек, который начинает работать над проектом, не знаком со всеми его деталями и уже существующими костылями.

В проекте нет документации, код не задокументирован, архитектура неочевидна, одни и те же вещи делаются в разных местах по-разному

1. CSS

Самое плохое, что может быть с CSS – вы до сих пор пишете CSS. Если это так, то немедленно ознакомьтесь с препроцессорами, узнайте об их плюсах и минусах, попробуйте и выберите тот, который подходит вам, или работа с которым попросту показалась вам удобнее. Нынешние препроцессоры имеют очень богатый функционал, ускоряют разработку. Однако есть и обратная сторона медали – с CSS препроцессором легче писать любой код, в том числе и плохой. В CSS появилась магия, и часто она используется с излишествами или без достаточно хорошо проведенного анализа – а стоит ли так вообще делать. Стоит ограничить использование фич препроцессора и выбрать, каким именно образом использовать ту или иную фичу.

Сила препроцессора очевидна в переменных, миксинах, условиях, функциях – если код можно улучшить и сделать его более читабельным/расширяемым/масштабируемым при помощи препроцессора – это стоит сделать.

Плохие и хорошие правила CSS

a) Частое использование !important

!important в коде должно быть как можно меньше, потому что это прямой путь к неподдерживаемому коду. Если нет другого варианта, кроме этого, то это серьезный повод задуматься, правильно ли сверстан элемент, для которого нужно юзать !important. Рядом с такими вещами лучше всего писать TODO, и описывать проблему, которая привела к такому.

b) Обращение элементам по селектору тега или id.

Нужно четко разделять границы использования классов и идентификаторов для элементов. Id может быть использован на странице только в двух случаях – либо это какой-то элемент формы, для которого существует соответствующий элемент label, связывающийся с элементом формы при помощи атрибута for (что можно обойти – спецификация разрешает такому элементу форму находиться непосредственно внутри элемента label, тогда связь определяется однозначно на уровне нахождения этого элемента формы без помощи атрибутов), либо это элемент страницы, к которому привязан например уникальный обработчик события.

Все остальные взаимодействия с JS, распространяющиеся на несколько элементов, когда удобнее сделать выборку по классу, должны использовать специально для этого зарезервированные префиксы классов.

На такие классы не должны навешиваться никакие стили, и, в свою очередь, внутри JS для обращения к группе таких элементов должен использоваться только этот класс.

Использование каскада либо селекторов по тегу должно сводиться к минимуму – для тегов это касается лишь оформления пользовательского контента на странице, если он имеется. Селектор по тегу также сложнее переопределить, и в дальнейшем, чтобы переопределить те или иные свойства элемента, нужно будет писать все более сильные селекторы.

c) Табы/пробелы и прочие оформительские мелочи. Должна быть согласована длина таба либо количество пробелов.

d) При группировке правил для нескольких селекторов каждый селектор записывается на новой строке

e) Перед открывающей фигурной скобкой всегда ставится пробел

f) Каждое CSS-правило пишется с новой строки

g) В конце каждой строки ставится точка с запятой, либо полное их отсутствие, если это позволяет препроцессор и всех разработчиков устраивает такой вариант

Тут подойдет пример с записью последнего правила в блоке без точки с запятой и проблемы с диффом при добавлении правил

h) После двоеточия, разделяющего название свойства и его значение, всегда ставится пробел.

j) Введение локальных переменных, если это позволяет написать более удобный, масштабируемый, расширяемый код

можно какой-нибудь пример про стрелку, например

i) Магические числа. Если в коде используется число, источник которого неочевиден, такие числа должны быть переведены в относительные величины (скорее всего, это можно сделать, грамотно переверстав структуру блока, заодно добавив блоку переносимости и масштабируемости), либо в крайнем случае описать в комментарии тут же, откуда взялось это число – это может быть сумма других значений в случае с box-model, либо просто неявное соответствие с другим свойством, определенным в CSS или же вовсе неопределенным.

k) Использование хаков запрещено. Хаки уже почти канули в лету вместе со старыми версиями IE. Куча хаков, которыми был наводнен интернет лет 5-7 назад, касались всего лишь разных уровней поддержки браузерами элементарных CSS3 селекторов. Все эти «хаки» уже давно стали неактуальными. Хаки для старых версий IE основаны на добавлении корневому элементу страницы некоего класса, который точно определяет версию IE – а в CSS лишь происходит каскадное переопределение либо добавление нужных свойств.

l) спрайты – Compass давно умеет собирать спрайты, сейчас для этого также есть плагины гранта. Если нет ничего на стороне автоматизации, нужно позаботиться об этом. Запросы, все дела.

m) комментарии в коде

Комментируйте свой код и особенно неочевидные решения или связи, если таковые есть, помечайте их TODO, описывайте причины, побудившие поступить именно так.

n) использование транслита

Английский язык де-факто стандарт в сфере разработки. Все именования должны быть на английском. Использование кириллицы крайне нежелательно.

--

Цените не себя в искусстве, а искусство в себе.

Имена классов в CSS не должны основываться на репрезентативном представлении элемента. Если сегодня кнопка синяя, это не значит, что завтра она останется синей, но информационный смысл такой кнопки скорее всего сохранится, потому что это логическое представление элемента. Следовательно, привязывать имена классов стоит к тому, что конкретно из себя представляет тот или иной элемент на странице и какую логическую роль он выполняет.

CSS reset. Не так важно, каким именно он будет – мейеровским, нормалайзом или самописным, или чудесной смесью всех трех вариантов – главное, чтобы выполнялись два условия: первое – он подходил вам для вашей разработки; второе – вы понимали, зачем в этом резете сбрасывается каждое свойство.

Сортировка CSS-свойств. CSS Comb. Зачем это – затем, что свойства должны быть рассортированы внутри блока правил согласно какому-то порядку. Но этот порядок ни в коем случае не должен быть логическим. Порядок должен нести в себе смысл – например, от более глобальных и важных свойств к менее важным и локальным.

CSS properties order within the rule

a) custom special props for element

('content' for pseudo elements)

b) props for definition the type and general positioning element

(e.g. 'display', 'float', 'position', 'left'/'top'/'right'/'bottom', etc.)

c) props related to box-model and dimensions of element

(e.g. 'box-sizing', 'width'/'height', 'margin'/'padding', 'border', etc.)

d) props for unique element

(e.g. 'list-style', 'table-layout', etc.);

e) misc props related to whole element's space

(e.g. 'background', 'opacity', 'overflow', etc.)

f) misc decor props related to whole element's space

(e.g. 'transform', 'transition', 'filter', etc.)

g) props related to inline/text content within element or to children' behavior

(e.g. 'vertical-align', 'text-align', 'white-space', etc.)

h) misc decor props related to inline/text content within element

(e.g. 'font-family', 'font-size', 'font-weight', 'text-transform', etc.)

Graceful degradation

1. Методология

На данный момент есть много методологий. Для чего они? Они не говорят вам, как стоит писать код. Они помогают писать код так, чтобы он легче поддавался дальнейшим изменениям.

Чтобы несколько людей, работая в команде, писали одинаковый код – это увеличивает производительность команды

Ознакомьтесь с основными методологиями, выберите себе по нраву, или соберите свое из частичек, как это случилось с OPOR.

4. Автоматизация фронтенда

CSS-препроцессоры, затем CSS-постпроцессоры, затем сборщики статики, затем пакетные менеджеры, grunt/gulp. Наконец, yeoman.

Чем меньше времени вы тратите на типовые рутинные задачи вроде сборки CSS, оптимизации картинок, тем больше времени у вас остается на что-то интересное – изучение новых технологий.

5. Общие проблемы и пути их решения

Есть много причин возникновения такого плохого кода.

Недостаток информации о продукте

Непродуманная архитектура. Здесь, мне кажется, причина кроется в том, что среди верстальщиков, особенно начинающих, существует общий слабый уровень знания основных, ключевых правил построения архитектуры, правильной модульности проекта. Основные и самые долгоживущие баг проекта закладываются в самом его начале.

Недостаток общения между различными людьми в команде (дизайнер - верстальщик) либо недопонимание между ними.

Отсутствие выделения времени под технический и рефакторинг в общее время разработки. Очень часто люди, распределяющие рабочие ресурсы в команде, считают, что проблемы у проекта есть на стороне сервера, мощностей серверов, возможностей админки, но не со стороны верстальщика. Хотя иногда именно по этой причине сдвигаются или срываются сроки релиза той или иной фичи из-за распутывания клубка в CSS с тем, чтобы не поломать другие части проекта.

Отсутствие личной инициативы и мотивации – ведь кому хочется решать скучные нудные задачи, ничего при этой не получая.

Статичная документация никогда не будет в состоянии up-to-date. Она будет постоянно догонять развивающийся код, обрастающий новым функционалом и архитектурой. Код должен сам себя документировать, чтобы, читая его, были понятны основные принципы его построения и его структуры.

Навигация в архитектуре и коде // Source Maps

Есть ли предел совершенству, когда можно сказать, что ваш код выглядит идеально? Думюа, никогда. Всегда есть что-то, чего нехватает, и это что-то может быть не всегда очевидно, и вообще не всегда может существовать, оно может стать необходимым лишь на какой-то стадии развития продукта. Стремление к этому идеалу и есть развитие продукта.

Именование классов – привязка к опыту

Микродата -