ТЕХНОЛОГИИ

Снижение количества женщин в отрасли из-за «резко» возросшей популярности универсальных сред – лишь одна сторона медали, затронутая автором. На мой взгляд, женский пол с лёгкостью способен разбираться в сложных математических абстракциях и работать в более универсальном пространстве, недели в коридоре узкоспециализированных программных сред. И ссылаться на половую принадлежность, как это делает автор, грубо.

Почему же тогда девушек меньше? Увы, к сожалению, далеко не каждую девушку в школе и дома обучают так, чтобы к университету у неё было не просто образно-гуманитарное представление мира, но и формализовано-техническое, которое позже позволит ей стать специалистом в IT-сфере и без труда справляться с теми проблемами, с которым справляются мужчины.

На дворе 2015-й год, и мной одолевают очень сильные сомнения, что за пределами университетов есть курсы и / или предприятия, которые пишут рабочий и окупаемый продукт, не используя Git или другие ревизионные системы.

Хочется по своей наивности верить, что далеко не все заказчики представляют из себя «большой глупый кошелёк», как это с одной стороны было продемонстрировано в главе «Диалог о производительности». Уверен, что есть компании, которые «разрабатывают сами на себя» (к примеру, EPAM и гиганты вроде Google, Microsoft, Apple), где заказчиком выступает человек из компании (CEO).

Битва фреймворков и узкоспециализированных сред разработки мне не совсем ясна. Точнее, не ясно, почему этот вопрос стоит так остро. Возможно, это оттого, что я сам использовал фреймворки для написания некоторых программ. Но на мой взгляд, надо чётко понимать, что где-нибудь в NASA или на военных разработках, они используют узкоспециализированный софт, чтобы проводить более точные расчеты. Не исключен также и вариант написания той же NASA своего программного решения (если хотите, то это будет фреймворк), которое позволит им сделать производительность повыше и точность побольше (зависит от исходной цели разработки данного продукта). Затем этим фреймворком могут воспользоваться другие компании, которым необходим аналогичный функционал и которым необходимо экономить время с деньгами на разработку. Плохого в том, чтобы не тратить время на «создание велосипеда», я не вижу (только если это не университет, где написание фреймворка даёт пишущему чёткое анатомическое представление того инструмента и последующих, которые он будет использовать).

Не понравилось, что веб-разработка автором едко задвигается в далёкий тёмный угол, дабы не мешала разрабатывать и развёртывать настоящие полноценные софтины для декстопников и тому подобных спецов. Складывается не очень хорошее впечатление, что автор просто не ужился с новой ~~женой~~ технологией. Если не можешь мириться с тем, что времени стало уходить больше на какие-то мизерные проблемы, то пора уходить из отрасли. Хотя и опасения автора тоже понятны, но, скорее, с точки менеджера: времени действительно стало уходить больше на всякую «мишуру» и «конфетти», особенно это видно в чётком разграничении должностей и сфер, типа «разраб» и «дезигнер».

Как ни крути, но, увы, нельзя вот так просто взять и забыть о сфере веб-разработки даже на микросекундочку, ибо рынок уже как с 2008 года дядюшкой Стивом Джобсом построен и заточен под мобильные технологии, которые направлены на максимальное удобство и (что вполне логично, исходя из названия) мобильность. Есть тут и подводные камни в плане безопасности пользователя, но, как оказалось, их предостаточно и у разработчиков, которым эта технология была в новинку. Об этих камнях автор достаточно подробно рассказал.

В утешение будет сказано, что софт для десктопов и суперкомпьютеров до сих пор спонсируется и разразабатывается, изменилось лишь отношение к нему, как, впрочем, и к стационарным ПК с ноутбуками. Всем захотелось лёгкости и мобильности. А также воздушности и облачности. Посему для ПК/ЭВМ/техники разрабатываются либо какие-нибудь профориентированные штуковины (банк.софт для ATM, игрульки на XBOX и т.д.), либо урезанные игрушечные шарики с рождественской ярмарки веб-серверов и веб-клиентов.

Благо сейчас время действительно другое (бежит уж очень быстро; книга написана в 2013м, а на дворе заканчивается 2015й, по прикидкам, информация успела возрасти в раз 16 – в два раза за каждое полугодие). Да и получение знаний благодаря курсам уже не кажется таким кощунственным как и сама веб-технология со всякими дебрями вроде скриптовых языков, разных браузеров и развёртываемости на всех устройствах от планшета до микроволновки. Можно точно сказать, что ситуация стала обратной: с вебом многие уже на «ты», а вот как софтину написать на плюсах знают только бородатые «битарды» с пивным от опыта животом.

Изменится ли что-нибудь в ближайшее время?

Да, если технология начнёт массово давать сбои и ставить нас в неловкое положение из-за скрытых и / или забытых уязвимостей самого веб-программирования. И нет, если веб-разработку доведут таки до ума.

Всё меньше и меньше начинает нравится подход автора к новым веяниям, начинает чувствоваться субъективная неспособность перестроиться. Конечно, это, с одной стороны, даёт массу тем для дискуссий, с другой стороны, формирует однополярное мнение, какую-то избитую категоричность, из-за которой проблема хоть и разобрана, но только с одного боку.

Прежде всего, меня возмутил тот факт, что в глазах автора практически вся молодёжь не знает SQL. Конечно, у него опыт больше, чем у меня, да и людей и фирм он повидал больше, но, на мой взгляд, сейчас обстановка более-менее налажена, тот же конвейер и даже курсы уже следят за тем, чтобы выпускаемый полуфабрикат был закалённым, чтобы знал азы вроде Java / C++ / SQL / JScript / HTML.

Понимаю и соглашусь в то же время, что рынок программистов находится в странном состоянии, а именно: людей не хватает, но полуфабриката тьма. В принципе, если только таким взглядом и смотреть с ~~авторской колокольни~~ позиции автора, то ясно его негодование. Да, супер-пупер специалистов-профессионалов готовят исключительно в вузах, и то там есть свой отбор-сито, через которое проходят далеко не все.

Но, на мой взгляд, это уже проблема человека, что он чего-то не знает или не прошёл отбор. Или на курсы не попал. Плюс никто не отменял такого правила: кто хочет, всегда сможет. Так работает Вселенная, по такому закону работаем и мы.

Касательно позиций насчёт ООП тоже согласен не полностью, ибо паттерны для того и созданы, чтобы облегчить построение более сложных систем. Да, правильно было замечено, что во многом, если не во всём, они представляют кальку с природы, перенесённую на связи и принципы ООП. Опять же, мы довольно ограничены в технологии (а может и фантазии), чтобы воплотить в жизнь мечту автора (и не только его): сделать связи красивее, элегантнее и настолько проще, чтобы можно было не запутываться в построении сложных систем. Пока что никуда мы не денемся от понимания на уровне паттернов, однако, я считаю, что не за горами языки или системы, в которых реализация паттерны будет проводиться так, что нам не нужно даже знать особую структуру в зависимости от случая, просто написать пару инструкций.

Насчёт СУБД и ООП во многом понимаю автора, но должен сказать: таковы правила, надо по ним играть. Это здорово, когда программист знает SQL и запросто может написать крутые запросы, но, на мой взгляд, коль мы крутимся в сфере веб-разработки, где всё сплошь и рядом одни объекты, то, будьте так добры и любезны, также уважительно относиться к ООП и как следствие оборачивать взаимодействие объектов / моделей с базами данных в ORM.

Это не значит, что не стоит придумывать новые схемы. Стоит. Но, как очень важно подмечено Тарасовым, необходимо наличие математической теории, которая это дело позволяет обобщить. А это важно, в особенности, когда размеры растут и необходимо искать какие-то закономерности.

Ещё позабавило во многом отношение автора к методологии разработке SCRUM. Вот уж чего я не ожидал, так этого такого едкого сарказма! Да и в адрес весьма популярной штукенции, которую никто мимо не пропустит! Да, во многом это верно, ведь это затормаживает процесс, добавляет больше пустословия и бюрокатии, которая продукт к завершению не движет. Да и денег, я уверен, на этот самый «стыд-и-скрам» тратится больше, чем если бы ребята им не пользовались.

Конечно, это дело вкуса и во многом престижа. И, на мой взгляд, то, о чём кричит Сергей Тарасов в этой главе, так это о том, как напускные аспекты вроде того же престижа, влившиеся в понятие «разработки программного обеспечения», изуродовали отрасль, в которой начали прорастать очень ценные зёрна (ООП, СУБД, паттерны и т.д.).

Взгляд, конечно, весьма консервативный, но он определённо заслуживает внимания.

**ПЕРЕЛОМНЫЙ ПЯТЫЙ КОНСПЕКТ**

**ВКЦП в облаках**

**Вычислительная мощность ПК не используется на 100%.**

Неудивительно, ибо рядовой пользователь выполняет с помощью ПК достаточно нехитрые задачи. В тех случаях, когда он выполняет что-то сложное, что нагружает ЦУ и ОЗУ, то, во-первых, стоит задуматься, а является ли рядовым данный пользователь (может быть, это программист вроде Константина Антоновича Зубовича, которому захотелось «поиздеваться» над программой, используя её слабые места), и, во-вторых, делает это всё тот же рядовой пользователь неявно, с помощью вспомогательных средств. Это, в частности, относится к геймерам: они ведь не производят ручные расчёты на матрицах, не влезают в дебри линейной алгебры и теории операторов, чтобы получить ту картинку и тот функционал, который у них есть благодаря тому, что кто-то из числа оговоренных выше лиц (программистов и не только) уже со всей этим достаточно «накувыркался». Хотя, надо признать, игра, провоцирующая людей на дотошное копание ради мало-мальского результата для меня есть очень даже неплохая идея для развивающей игры (для студентов ПМ) ☺

**Из-за избытка мощностей на узлах (в случае распределённой системы) происходит переход от распределённых систем в пользу централизованных.**

И я считаю, что это правильно, что если есть излишки в системе, которыми никто не пользуется (опять-таки, надо понимать, когда это «никто» не такое уж и пустое множество). Далее автором книги будут приведены доводы в пользу централизованных систем, в частности, речь пойдёт о ВЦКП и её сути. Во многом, с этими доводами, которые как все дороги ведут к Риму в виде «ВКЦП первым пытался реализовать взаимовыгодную систему облачных вычислений», я согласен и считаю, что шаг этот, принятый и НИИ, и, частично, государством, был верный.

**ВКЦП = Вычислительный Центр Коллективного Пользования**

На мой взгляд, тесная связь с ныне популярными «облаками» очевидна. И для советской страны в 1972 году первым сделать такие прорывные попытки, при этом затронув важную связь между экономикой, государственной сетью и объединением региональных центров, было просто великолепным подспорьем, которое, глядишь, могло бы сделать много полезного шума и предоставить конечному пользователю (да и государству тоже, об этом не забывать) много удобств.

**Суть ВЦКП со временем не изменилась, сменилось лишь название.**

Согласен. Опять же, надо видеть в ВЦКП скрытую систему работы с центральным сервером-«облаком» и все вопросы и сомнения отпадут: да, это то, чем сейчас интересно заниматься очень большому количеству программистов. В том числе и мне интересны такие подходы.

**ВЦКП пролетело:**

1. **Профуканная миниатюризация**
2. **Просчёты и развал страны**

В точку. Я бы ещё добавил в этот список самое по себе понятие «лихие 90-ые», когда работать надо было за еду, когда во все сферы жизни стал вмешиваться рынок, отчего налаженная до того система «мороженное за 3 копейки» превратилась в чёрти знает что. Понятно, что рано или поздно в программирование примешалась бы экономика, расширив и размыв понятие, но то, что это у нас произошло так бурно и стремительно (без подготовки, как на Западе) ещё раз, на мой взгляд, доказало, что ~~новые русские не сахар~~ стоит уделять время прогнозированию будущих решений и всегда держать ухо востро.

**Возврат к централизованным системам?**

Возможен. И я даже знаю почему.

Это предлагает тем же геймерам, а значит, одной из самых «жирных» прослоек современного инфообщества неописуемые возможности, ровно как и прибыль таким корпорациям, как Sony, Nintendo, Microsoft. Если подробнее, то не секрет, что та же Sony ведёт разработки «игровых подписок» для конечного пользователя, что из себя представляет тот же централизованный подход, но уже по отношений к играм. Фактически, игрок будет играть в одну и множества доступных за определённую плату терминалом «игровых проекций», которые запущены / поддерживаются сервером-«облаком».

Плюс, эта система уже активно используется, вспомнить хотя бы банковское дело. Программисты в банковской сфере не только много получают за прогнозирование рынка, но и за создание таких централизованных систем и поддержку безопасности для терминалов, коими и являются небезызвестные банкоматы.

На мой взгляд, за этими подходами будущее. Но всегда стоит помнить, что даже у такой схемы есть скрытые слабые места как банальное падение серверов и, как следствие, пропадание всего функционала на терминалах. Как это обойти – время покажет.

**Затратность программирования распределяемого приложения, но его выигрышность в сравнении с централизованными приложениями.**

На мой неподготовленный взгляд, затраты и там, и там. Где больше судить не берусь, ибо толком не работал ни с одним из направлений. В плане выигрышности, на мой взгляд, стоит опять же вернуться к игровой сфере (Остапа Бендера понесло ☺) и взять, к примеру, мультиплеерные игры.

Тяжелы ли они для разработки? Ну да, надо попотеть над хорошим мультплеером больше, чем над обычной синглплеер игрой.

Интереснее ли пользователю мультиплеер или синглплеер? Здесь чёткого ответа нет, есть только статистика, которая показывает, что с поднимающимся уровнем технологий, используемых в игростроении, мультиплеер всовывается практически во все игры, а это значит, что да, пользователю это «необходимо». Если смотреть более объективно, то это дело извращённости и вкусов конечного пользователя, ему, во всяком случае, необходимо предоставить право выбора.

Так, основываясь на игровой индустрии, можно сделать выводы и обобщения для других сфер.

В заключение скажу, что вот в чём я точно уверен, так это в том, что разработка приложения, которое выборочно совмещает в себе и распределённость, и централизованность, очень сложный процесс, который нуждается скорее в какой-то теоретической подоплёке для дальнейшего упрощения, нежели в супер-росте технологий.

**Работа с локальной копией данных центральной БД.**

Решение отличное, особенно для банков, но, хотелось бы верить, далеко не единственное.

**SaaS и data manipulations**

SaaS = software as a service

**Хороший вопрос: «софтерные фирмы производят услуги»?**

Хороший вопрос на подискутировать.

Соглашусь с точкой зрения автора: услуги не производят, а ОКАЗЫВАЮТ. А фирма производит ПРОДУКТ, а не услугу. Разница между ними в том, что продукт можно хранить, он физически осязаем, и время и место его потребления могут растягиваться, чего не скажешь об услугах: оказал, и она тут же «нашлась к месту».

Если уже принять тот факт, что фирмы = продукт, то без схем, описанных автором, не обойтись.

1. Производитель создал продукт, продукт обязательно проходит по рукам дистрибьюторов (консультанты, продавцы и т.д.) до конечного пользователя.
2. Программа как услуга сразу доходит до конечного пользователя, минуя дистрибьюторов (единственный это сам Интернет). Если пользователь программного продукта работает с ним через ВЦКП, то последний предоставляет доступ пользователям к налаженной системе.

Из чего следует, что в первом случае «навар» и проценты приведут к излишней дороговизне продукта, который может быть не до конца отлаженным, а во втором случае есть риск краха всей системы ВЦКП. Но и надо отметить, что рынок добрался и до второго пункта: появилась цифровая дистрибуция, люди за это получают денег не меньше, чем за обычную, «аналоговую».

**От CORBA к SOA**

CORBA – Common Object Request Broker Architecture

SOA – Service Oriented Architecture

**Бурные 90-ые, шум вокруг понятий**

.. что автоматически приводит нас к уже ранее затронутой теме «почему облажались с ВЦКП». Так, отчасти, и тут, с CORBA и SOA. Конечно, рассудили здесь не «по понятиям», но по простоте и популярности.

**Интероперабельность CORBA**

Огого, поддержка одной архитектурой множества различных платформ и архитектур поменьше/повыше всегда меня восхищало и радовало. Не знаю, стоила ли эта игра свеч, если она в итоге выдохлась. Надеюсь что стоила, ибо судить о том, чего уже практически нет и с чем работать не придётся очень трудно и поверхностно.

**Почему CORBA улетела с рынка?**

1. **Долгий процесс согласования стандартов, отчего рождаются решения «на местах» (у Sun, к примеру)**
2. **Сложность кода (для Javистов)**
3. **Сложность спецификации + появившиеся упрощённые Java решения**
4. **Бум дот-комов и, как слдествие, дороговизна коммерческого решения**
5. **Раскрутка упрощённой реализации веб-сервисов в SOA**

По поводу пролёта с CORBA автор всё изложил довольно чётко, поэтому мне нечего добавить.

**Sun могла объединиться с CORBA**

И никто бы из двоих не улетел, все были бы в выигрыше и концепцию довели до ума. Да, согласен с этим.

**SOA появилась как хорошее, но сырое подспорье CORBA**

И её сырость заключалась, по словам, автора, в вынужденных надстройках для достижения базового для CORBA уровня и простоты.

Зачем тогда необходимо было пересаживать всех и вся на сырую, но новую землю – я не могу понять. Возможно, было просто дешевле, а это, как на зло, не такое уж и программирование, чистый маркетинг, поднасоливший индустрии. Или, быть может, не всё так прозрачно?

**Из-за упрощения в SOA «порезали» функционал CORBA, а именно:**

1. **Отсутствуют сессии и состояния в вебе**
2. **Переход от интерфейсов на C к XML.**
3. **Отказ от автоматической подгрузки связанных объектов (lazy initializing)**
4. -> отсутствует возможность транзакций -> из-за чего надо использовать шаблоны (ВСЕГДА И ВЕЗДЕ) -> more code, MORE CODE!

У серверов нет состояния -> нет и объектов

1. \*(Ох, XML ужасен, ужасен, ужасен)\*
2. Тоже глупо, ведь плюсов у ленивой инициализации хоть отбавляй.

**Плюсы CORBA: хорош, как многопоточный сервер, хорош как веб-служба.**

**Минусы CORBA: см. пункты выше, точнее, почему CORBA вылетела**

**Плюсы SOA: затмеваются минусами**

**Минусы SOA: сырьё, недовведенное до ума**

Хотелось бы верить, что нас ждут другие решения.

**Прогресс неотвратим**

**Вымирание больших ЭВМ**

А где же ЦОДы и «облака»?

Большие ЭВМ не вымирают и не вымрут никогда, ибо они являют собой образцы стратегичности, требовательности в купе с защищённостью и производительностью. А за это многие дядьки с тугими кошельками отдадут денюжки.

Очень понравилась мысль: **«Солдаты от софтостроения не должны рассуждать, ибо генералы договорились с поставщиком».**

**Где тогда грань между рабским трудом и профессией программиста?!**

**.NET**

Из-за постоянных изменений, так активно производимых Microsoft (они там определиться не могут, или решают зарабатывать на «сыром и вялом»), .NET недозрелая платформа: концепции меняют как перчатки, а это несёт убытков больше, чем дохода (курсы и т.д.). Однако, надо сказать, что тому же Microsoft это, наверное, выгодно, предоставлять возможность переобучения для новой платформы (т.е. всё-таки доход, но для Microsoft, а для компании, которй нужно сотрудников переобучить, это убыток).

**Office ‘07**

Нет вшитой поддержки старых версий и нету возможности выбора интерфейса (если только не проплатить). Да, я помню, лично на себе испытал эти проблемы и скажу, что Word 03 самый стабильный и лучший из MS OFFICE пакетов. LibreOffice не в счёт, всё-таки за него не надо платить и этот «леденец» пришёл ко мне с опытом, отчего я его ставлю, как и автор, выше остальных.

**SQL Server**

Деградация: унификация, торможение из-за .NET, упущение мелочей из старых версий, несовместимость форматов.

**Vista**

Потренировались на вас за ваши деньги – в точку!

Продажа долей Гейтса и Балмера – хм, хм, хм. Что бы это могла значить.

**О материальном**

X64 плох, драйверов на него нет и он отрезает путь к использованию старый версий и старого софта.

Есть немного, но поскольку уже 2015-й год, с этой проблемой уже разобрались и драйверов хватает. Другое дело, что ставятся они зачастую некорректно и нужно ручками это исправлять. Приятно ли это? Кому как. Мне, допустим, да, приятно, я люблю копаться, я от этого больше понимаю. А вот обычный пользователь по логике вещей пошлёт систему на три буквы.

И посылает. Но продолжает пользоваться ей в силу натренированной беспомощности взять и пересесть на MAC OSX / Linux.

ПРОЕКТИРОВАНИЕ И ПРОЕКТЫ

**Проектирование и процессы**

**Совершенство достигается не тогда, когда нечего добавить, а тогда, когда нечего убрать. (Антуан де Сент-Экзюпери)**

Великолепный эпиграф. Эти слова без сомнения должны быть девизом уважающего себя проектировщика.

**КИС = Корпоративные Информационные Системы**

**КИС претерпели развитие от закрытых монолитов до открытых / модульных систем, вместе с тем пожертвовав стандартизацией процессов и базовой функциональностью ради конкурентоспособности.**

Ах, рынок, что же ты творишь?!

**Краткий словарь начинающего проектировщика**

Самый полезный (и весёлый) параграф из всех в этой главе

**Задача проектировщика – поиск простоты.  
Очень просто сделать сложно. Сложнее делать проще.**

**Сам словарик (не нуждается в комментариях):**

• «Это был плохой дизайн». Это спроектировано не мной.

• «By design» (так спроектировано). Ошибка проектирования, стоимость исправления которой уже сравнима с переделкой части системы.

• «Это не ошибка, а особенность (not a bug but a feature)». Прямое следствие из «by design».

• «Это может ухудшить производительность». Не знаю и знать не хочу ваши альтернативные решения.

• «Нормализация не догма». Потом разберёмся с этими базами данных, когда время будет.

• «Это наследуемый модуль». Этот кусок со многими неявными зависимостями проектировали достаточно давно, скорее всего стажёры.

• «Постановка задачи тоже сложна». Ума не приложу, откуда возникли эти десятки тысяч строк спагетти-кода.

• «Сроки очень сжатые». Мы давно забили болт напроектирование.

• «Наши модульные тесты покрывают почти 100 % кода». А функциональными тестами пусть занимаетсязаказчик.

• «В нашей системе много компонентов». Установку и развёртывание системы могут сделать только сами разработчики.

**Слоистость и уровни**

**Разбираться в КИС надо не только разработчикам самой КИС, но и программистам, которые собираются что-то «наращивать» на этой системе.**

**АИС = Автоматизированная Информационная Система.  
Состоит из:**

**Людей**

**Информации**

**Компьютеров**

**Устройство АИС с точки зрения проектирования делится на:**

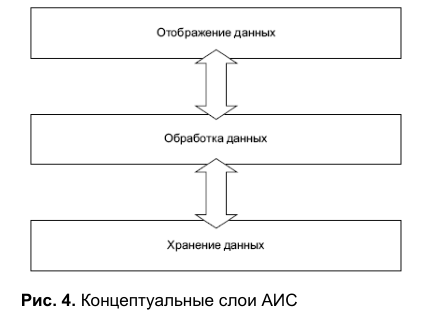
**Концептуальное устройство**

**Логическое устройство**

**Физическое устройство**

**Концептуальное устройство**

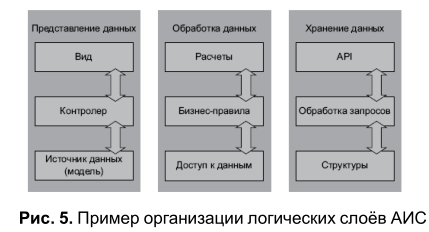
Используется анализ для составления устройства.

****

**Представленные слои существуют в ЛЮБОЙ ИС. Просто нужно уметь их искать.**

**Логическое устройство**

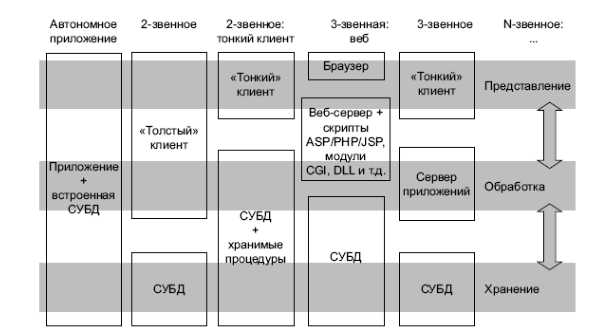
Используется синтез для составления устройства.



Нет строгого определения. Поэтому определение организации происходит путём ответа на вопрос «зачем нужен этот слой» либо использованием шаблонных решений и их подгонки.

**Физическое устройство**

Используется синтез для составления устройства.

****

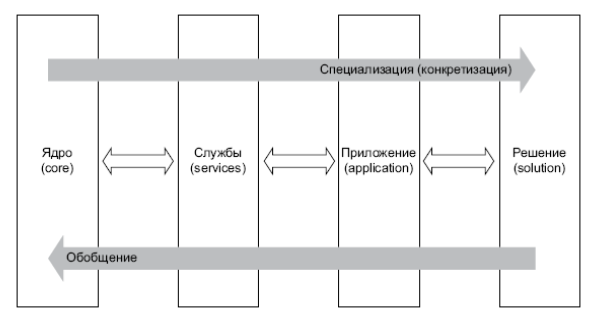
**Thin Client – имеет только логику отображения (терминал / браузер)**

**Rich Client – прикладная обработка логики без сервера, сервер как хранилище**

**Smart Client – золотая середина между Thin и Rich клиентами**

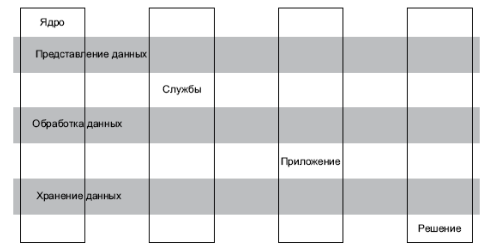
**Насыщенное веб-приложение – Thin оснастка + возможности Rich**

**Уровни**

****

**Общие функции вынося в модули / API системного уровня.  
Функционал нескольких предметных областей выносится на уровень решения.**

**Совмещение**

****

**Многозвенная арихтектура**

**В концептуальном устройстве всего три части.**

**В физическом их может быть сколько угодно.**

**Доля специалистов по системному программированию падает -> архитектуру выбирают особо не думая, прочитав статью и не разобравишсь, либо «потому что иначе не умею».**

В этом плане я вспоминаю Константина Антоновича Зубовича и его великие слова: «Что вы, Гетьман, по помойкам разным лазите? Лучше бы книжки читали». Само собой разумеется, что под помойками понимается СМИ и неподтверждённая информация, такая как заметка на блоге / форуме или пущенный слушок.

Как с этим бороться? Воспитывать интерес самообучения в людях, основанный на недоверии к глупостям, незнанию, поспешным и необдуманным решениям.

**Прослойка парсинга информации достигает 80% от всей работы приложения, а ведь это пустой процесс.**

Да, это пустой процесс, пока вы пользуетесь Google, который активно обрабатывает такие big data, что вам и не снились.

**Практики управления «кейсами» в ИТ…**

… и русифицированный клон западного приложения? Или же ещё один успешный проект?

Смотря, кто и как использует эти «кейсы». Если менеджер толковый и не зря ходил на лекции в студенческие годы, то он воспримет всё это как совет в рамках одной глобальной игры под названием «как утереть товарищу нос», ведь кейсы ничем не отличаются от популярных технологий: всплывают тогда, когда больше не нужны. Ну а если менеджер глуповат, то будет всё делать по «инструкции» и сам проект становится каким-то безжизненным.

**Критичность проектов в ИТ падает, отчего решения переходят из технической в управленческую сферу.**

**Ищите возможности сократить путь информации от источника к пользователю и обратно.**

**Архитектура не есть освоение бюджета плюс вовлечение сомнительного персонала.**

100% верно. Эти слова да всем чиновникам в уши.

**История нескольких #ifdef**

КИС, реализующая основные функции автоматизации деятельности торгово-производственной фирмы среднего размера: от бухгалтерии и складов до сборочного производства и сбыта – была разработана командой из 4–5 человек примерно за полтора года, включая миграцию с предыдущей версии. Система критичная, даже короткий простой оборачивается параличом деятельности фирмы.

Причина столь сжатых сроков? Ясное понимание решаемых прикладных задач, создание соответствующего задаче инструментария, прежде всего, языка бизнес-правил высокого уровня, и подтверждение тезиса Брукса о многократно превосходящей производительности хороших программистов по сравнению с остальными.

**Ultima-S – КИС из коробки**

**Техническая культура – это не производства и знания, а люди, умеющие это делать и применять.**

Можно и мартышку научить «мыслить шаблонами», можно и в роботов вложить необходимые конвейерные знания. Но всегда нужны будут те, кто будут дрессировать мартышек, создавать роботов и мыслить «выходя за плоскость».

**Два похода к разработке КИС: «от производства» (надо распланировать и оптимизировать имеющиеся ресурсы производства) и «от бухгалтерии» (надо проанализировать хозяйственные операции и на основе полученных данных спрогнозировать дальнейшие действия).**

Думаю, здесь автор всё достаточно подробно и чётко изложил.

**Проектированием системы на самом деле должен заниматься не технарь, а маркетолог как Стив Джобс. Система – это товар, удовлетворяющий потребности. Маркетолог может понять потребности рынка, сформулировать требования к товару, может оценить осуществимость маркетинговой компании. На деле технический архитектор нужен маркетологу для оценки себестоимости проекта и его сроков, а также для информации о новых перспективных технологиях, тогда маркетолог сможет искать сочетания «потребность + технология».**

Всё уже сказано, дополнить мысль нечем.

Однако, тогда мне не понятно за счёт чего первое время держатся стартапы, ведь у них в команде даже толком менеджера нет, я уже и не говорю о маркетологах. Конечно, с одной стороны, я, возможно, слишком идеализирую образ стартапщика, этакого молодца, в одиночку сворачивающего горы с гордо поднятой головой, но, тем не менее, нельзя исключать и такого развития событий, где команда из трёх технарей выстреливает на рынке за счёт пробивной идеи. Наверное, я сам и отвечу на свой сомнительный вопрос: команду, как и проект, в таком случае кто-то более солидный и опытный замечает, а он то, в свою очередь, и подыскивает ребятам, которые не сильны в финансовой грамоте (весьма и весьма условное ограничение) и бизнес-аналитиков, и менеджеров, и маркетологов. А они уже делают своё дело, которому их также как и наших умельцев-технарей обучали.

**Следуйте совету Джобса: «Обычные художники заимствуют – великие воруют».**

Очень и очень спорные слова. Не могу прямо сейчас понять, что же мне в этой фразе не нравится: то ли тот факт, что эти грязные слова были произнесены довольно авторитетным человеком, то ли действительная отрезвляющая правота этих слов, сбивающая мои «розовые очки». Трудно сказать.

Но, стоит заметить, что если эту цитату немного смягчить и развернуть под углом, мол, смотрите на других и берите у них самое лучшее, то я, безусловно, соглашусь. Это даже с психологической стороны полезно. А с технической, на мой взгляд, это одна из вещей, воспитывающая техническую культуру.

**Нешаблонное мышление**

**Выдумывание велосипедов**

Зачастую, ещё вчерашние новички, научившись достаточно элементарным вещам, любят порассуждать о том, что изобретение велосипедов – пустое дело. По моим представлениям на воспроизведение большинства из приводимых в книжке «велосипедов» в конкретных случаях у любого специалиста с навыками абстрактного мышления вряд ли должно уходить более часа. Это заметно быстрее изучения самой книги, ее осмысления и, наконец, осознания тех моментов, когда, собственно, надо применить абстрактный слепок в реальной жизни.

Сложнее обстоит дело с теми, кто только начинает свой путь. Прочтение сего труда новичком, как мне кажется, является прямым аналогом попадания в прокрустово ложе диктуемой парадигмы. Потому что книга описывает набор решений, а неокрепшему за недостатком практики уму проектировщика надо научиться самому находить такие решения и пути к ним. Для чего гораздо эффективнее первое время «изобретать велосипеды», нежели сразу смотреть на готовые чужие. На чужие надо смотреть, когда придуман хотя бы один собственный, чтобы понять, насколько он несовершенен, и выяснить, каким же путём можно было бы прийти к лучшим образцам велосипедов данной модели.

**Thinking in patterns**

Несколько позднее я наткнулся в магазине на книгу с названием, претендующим на звание наиболее абсурдного из встречавшихся. Оно звучало как «Thinking in patterns». В переводе на русский язык – «Мыслить шаблонно».

Ещё недавно привычка шаблонно мыслить считалась в инженерном сообществе признанием ограниченности специалиста, наиболеепригодного для решения типовых задач с 9 утра до 6 вечера. Теперь не стесняются писать целые книжки о том, как научиться шаблонному мышлению…

**Думать головой**

**Наследование = обобщение**

Откажитесь от термина «наследование», который искажает смысл действий. Мы обобщаем.

Обобщаемые классы должны иметь сходную основную функциональность.

Основная ошибка – обобщение по неосновному функциональному признаку.

Ошибка приводит к построению иерархии по неосновному признаку. Когда внезапно найдётся ещё один такой признак, возможно, более существенный с точки зрения прикладной задачи, обобщить больше не удастся: придётся ломать иерархию или делать заплату в виде множественного наследования, недоступного во многих объектно-ориентированных языках. Или пользоваться агрегацией.

Не увлекайтесь обобщением. Ошибки тоже обобщаются и уже в прямом смысле этого слова наследуются. Исправление по новому требованию может привести к необходимости сноса старой иерархии, содержащей ошибки.

**Эмпирическое правило**

Глубина более двух уровней при моделировании объектов предметной области, вероятнее всего, свидетельствует об ошибках проектирования.

**Освобождение памяти**

Наиболее очевидное преимущество – программисту не надо заботиться об освобождении памяти. Хотя при этом все равно нужно думать об освобождении других ресурсов, но сборщик опускает планку требуемой квалификации и тем самым повышает массовость использования среды. Но за все приходится платить. С практической стороны недостатки сборщика известны, на эту тему сломано много копий и написано статей, поэтому останавливаться на них я не буду. В ряде случаев недостатки являются преимуществами, в других – наоборот.

Черно-белых оценок здесь нет. В конце концов, выбор может лежать и в области психологии: например, я не люблю, когда компьютер пытается управлять, не оставляя разработчику достаточных средств влияния на ход процесса.

**Журнал хозяйственных операций**

**Дам совет начинающим: учите сиквел, транзакции, уровни изоляции, и будет ваша система быстрой и надёжной.**

**UML и птолемеевские системы**

**В UML должно настораживать уже самое первое слово – «унифицированный». Не универсальный, то есть пригодный в большинстве случаев, а именно унифицированный, объединённый.**

Всё верно. Если абстрагироваться, то можно объединять много разного нехорошего и неполезного. От этого мы результат хороший не получим, сколько бы мы не надеялись на то, что «минус на минус даст плюс» или что какая-то мелочь затесавшаяся не вылезет на поверхность. Является ли подобным объединением UML? Нет, во всяком случае, уверен, есть задачи, когда использование UML облегчает производственный процесс или процесс разработки.

**Графические образы – это, конечно, аналог слов, но не всякий набор слов является языком. Поэтому UML до языка ещё далеко. Язык, даже естественный, можно автоматически проверить на формальную правильность. Входящий в UML OCL хоть и является языком, но выполняет лишь малую часть работы по проверке, и не собственно модели, а её элементов.**

**Очень простой, но верный признак проблемы – отсутствие в инструменте моделирования команды «Проверить». И если условной кнопки «Check model» в панели не предусмотрено, то ваш набор инструментов и методов для проектирования программного обеспечения является просто продвинутым редактором специализированной векторной графики с шаблонами генерации кода.**

**Итак, UML:**

**• не универсальный, а унифицированный, объединяющий отдельные практики;**

**• не язык, а набор нотаций (графических);**

**• не моделирования, а в основном рисования иллюстраций, поясняющих текст многочисленных комментариев.**

**У Киплинга есть замечательный рассказ «Как было написано первое письмо». Про неудачный опыт использования графической нотации первобытными людьми. Но если в рассказе все закончилось хорошо, то в современном проекте была ситуация, когда двум проектировщикам по отдельности поручили делать диаграммы переходов для одного и того же набора классов. Получилось очень «унифицированно»: найти хотя бы одну пару общих элементов в двух диаграммах было затруднительно. Если методология проектирования действительно существует, то работа двух людей даёт сопоставимый результат.**

**Это [UML] идеальный инструмент для создания птолемеевых систем. Только если модель Птолемея является геоцентрической, то прецеденты использования – модель заказчико-центрическая.**

Очень красивое высказывание, надо взять на заметку: «заказчико-центрическая модель».

**Слишком часто формулировка «лучше плохой, чем никакой» стала широко применяться, что настораживает.**

Поддерживаю.  
На днях была опубликована статья в онлайн-журнале TheJournal (ссылка: <https://tjournal.ru/p/everything-is-buggy> ), тема которой, если вкратце, то была такая: багги из приложений и опрерационных систем никуда не исчезнут, придётся с этим смириться, уважаемые граждане. Посему, вспоминается эта статья при чтении формулировки.

**Для тиражируемых продуктов и ориентированного на специализации проектного софтостроения наиболее интересен подход с разработкой собственных языков, где необходимость использования UML неочевидна. Остаются, по большому счёту, заказные проекты без видимых перспектив повторных внедрений, где само слово «моделирование» может вызвать реакцию «пиши код, мы тут проектируем только снизу-вверх».**

**К великому счастью, программисты системы не занимались тогда «эволюционной» разработкой, а знали область, в которой работают, и добросовестно поддерживали проектную документацию в актуальном состоянии.**

Не могу сказать, что сейчас прямо таки всё-всё-всё не так. Да, то, что раньше было узким, сейчас расширело, набрало массовость, либо исчезло за ненадобностью. Но, тем не менее, необходимость разбираться и делать это самостоятельно никуда не делась. Это одна из первых вещей, на которые смотрят во время испытательных сроков более опытные дяденьки и тётеньки. Да и сами опытные дяденьки и тётеньки на то и гордо носят название «опытные», что знают область не понаслышке, а по работе вживую. А ценнее этого никакая философия быть не может.

**Что имеем по итогам более чем 10-летнего развития технологий? Появилась возможность стандартизировать хранение и доступ к большому массиву данных, используя вполне обычное серверное оборудование корпоративного класса и 64-разрядную промышленную СУБД. Возникла необходимость переделывать программное обеспечение из-за утраты поддержки среды разработки поставщиком и соответствующих компетенций на рынке труда.**

**Но я совсем не уверен, что ещё через 10 лет новые подрядчики, вынужденные в очередной раз переписывать систему, найдут документацию в том же полном виде, в котором нашли её мы. Если вообще найдут хоть что-то, кроме кода, остатков презентаций и наших отчётов.**

**«Оптисток», или распределённый анализ данных**

**Наибольшую трудность, кстати, вызывает не сам импорт данных, а поиск в компании людей, которые могут знать, где эти данные взять. Всё-таки 20 тысяч таблиц SAP R3 и примерно столько же в корпоративном хранилище – не шутка, поэтому без хороших проводников раскопки источников информации обернутся поиском иголки в стоге сена.**

**Архитектура сокрытия проблем**

Просто история о том, как «разрабы» скрывали свои огрехи, основанные на пересмотре огрехов других. Не более того.

**Code revision, или Коза кричала**

**Хуже, когда вполне программистский коллектив умудряется годами работать без системы контроля версий исходников, и тогда в коде половину объёма составляют закомментированные куски многолетней давности. Выбросить их жалко, вдруг пригодятся. Но и контроль версий с архивацией не спасает от цифровой пыли десятилетий. В подобных залежах порой можно обнаружить настоящие образцы софтостроительных антипрактик.**

Уж увольте, но я не могу понять этого высказывания. Может быть, я в силу наивности так считаю, но, выгляньте в окно, на дворе 2015й год заканчивается, Git с Mercurial были придуманы лет десять назад и получили очень и очень широкое распространение среди программистов разного покроя и пошива. Даже на курсах людей учат пользоваться системами контроля версий.

Намотав на ус всё вышесказанное, взгляните ещё раз бегло на авторский текст. Что это за «вполне программистский коллектив»? Ребятки пишут софт на коленке для шарашкиных конторок? Если только так, ибо компании покрупнее, у кого в штате есть хотя бы один человек, который сечёт в чём-то кроме программирования, смекает, что надо бы как-то эти залежи поудобнее разгребать. Да и стартапщики сейчас не такие уж и забулдыги, чтобы не пользоваться СКВ. Напротив, это модно и приятно пользоваться системами контроля версий.

Насчёт архивации древних записей не могу толком ничего сказать, ибо не сталкивался с этим вплотную. Однако уверен, что гиганты вроде PreForce, Atlassian, BitBucket, GitHub, Mercurial что-то имеют по этому поводу. По крайне мере, это «что-то» достаточно хорошо сделано, ибо на том же GitHub можно найти очень и очень старенькие файлы, даже проектики, созданные в день открытия системы. И само собой, о выбросе старых исходников речь заходит только тогда, когда одна концепция / парадигма / один инструмент сменяются другими.

**Ревизия кода, несомненно, весьма полезная процедура, но как минимум при двух условиях:**

**• эта процедура регулярная и запускается с момента написания самых первых тысяч строк;**

**• процедуру проводят специалисты, имеющие представление о системе в целом. Потому что отловить бесполезную цепочку условных переходов может и компилятор, а вот как отсутствие контекста транзакции в обработке повлияет на результат, определит только опытный программист.**

Полностью согласен с автором. Зелёненьким проггерам или просто талантливым, но неопытным ребяткам доверять топор ревизии в руки нельзя, они начнут смущать тех, кто ещё зелёнее.

Я бы ещё добавил помимо всего прочего с первых строк кода делать прогонку на тестах, чтобы и guideline языка соблюдались, чтобы и функционал был работающий, чтобы вообще ничего и нигде е смущало и не падало. Обычно, на нормальных проектах так и делают.

**Большой ошибкой является привлечение к процессу внутреннего тестирования и обеспечения качества посредственных программистов. По его мнению, компетентность специалиста в этом процессе должна быть не ниже архитектора соответствующей подсистемы. Действительно, ведь оба работают примерно на одном уровне, прост один занят анализом, а другой – синтезом.**

И не поспоришь!

**Наживулька или гибкость?**

**Очень важно отделить редкую ситуацию «бизнес меняется еженедельно» от гораздо более распространённой «представления команды разработчиков о бизнесе меняются еженедельно». Если вам говорят о якобы часто изменяющихся требованиях, всегда уточняйте, о чём, собственно, идёт речь.**

Да.

\*\*\* О, да в этой главе рассказывается про спиральную методологию разработки! \*\*\*

**Ключевой особенностью гибкой методики является наличие мифологического титана – владельца продукта (product owner), который лучше всех знает, что должно получиться в итоге. На самом деле это просто иная формулировка старого правила «кто платит, тот и заказывает музыку». Именно владелец, за рамками собственно гибкого процесса, гением своего разума проводит анализ и функциональное проектирование, подавая команде на вход уже готовые пачки требований. Размер пачки должен укладываться в интеллектуальные и технологические возможности разработчиков, которым предстоит осуществить её реализацию за одну итерацию.**

Согласен.

**Методология, по сути, направлена на увеличение времени работы с клавиатурой и не располагает к размышлениям. Пиши код! Поэтому для стимуляции персонала процесс окружен религиозной атрибутикой, манипуляциями, иносказаниями и метафорами. С другой стороны, требования к уровню программиста ограничиваются знанием конкретных технологий кодирования, стандартных фреймворков, «умением разбираться в чужом коде» и «умением работать в команде», уже упоминавшимся в словаре для начинающего соискателя. Способность решать олимпиадные задачки здесь от вас не требуется. Скорее, наоборот, будет помехой.**

**Позитив для заказчика в том, что, осознавая свою несхожесть с мифологическим титаном мысли, он может достаточно быстро увидеть сформулированные требования и сценарии в реализации, отлитыми, разумеется, не в бетоне, а в гипсе, и на практике понятьих противоречивость и неполноту. После чего он может переформулировать существующие и добавлять новые требования с учётом уже набитых шишек. Тем не менее с ростом сложности системы возрастает и риск увеличения стоимости внесения изменений. И если проект выходит за рамки бюджета, то «козлом отпущения» становится именно владелец продукта.**

**Другой позитив для компании-заказчика состоит в непосредственной близости выполняемой подрядчиком работы. Зачастую «гибкие» команды работают на площадке компании и доступны в любой рабочий момент. Если заинтересованному лицу не хватает информации, он может просто подойти и посмотреть на месте, поговорить с исполнителем и тем самым восстановить расстроившееся было душевное равновесие.**

Комбо «согласен» x3.

**Позитив для подрядчика состоит в том, что «гибкая» разработка позволяет вовлечь в проект как можно больше разработчиков с менее высокими требованиями к квалификации. Это позволяет содержать больше сотрудников в штате, включая оффшорные команды. Будучи поставленным перед выбором между небольшой программистской фирмой с квалифицированным персоналом и софтверхаузом-«тысячником», крупный заказчик в общем случае склонится ко второму варианту.**

Логично. За счёт гибких методологий и живут компании. А вы как думали, верили в добрый исход и победу на рынке узкоспециализированных команд?

**Заказчик осознаёт, что реализовать спецификации своими силами невозможно, прежде всего потому, что при таком объёме они тем не менее неполные и неизбежно содержат противоречия. Подрядчику же в принципе наплевать на спецификации, он будет крутить итерации, честно реализуя заявленный функционал и отрабатывая бюджет. Получился коровник на подпорках с покосившимися заборами и дырявой крышей вместо современного агрокомплекса? Извините, всё по спецификации, каждые две-три недели вы видели расцвеченные фотографии разных участков возводимого сооружения.**

Без комментариев.

**Тесты и практика продуктового софтостроения**

**При изменении спецификаций затраты времени на приведение в актуальное состояния тестов могут быть куда больше, чем на собственно код. Скажем, если пользователи захотят поменять синтаксис SQL в СУБД и писать SEARCH вместо SELECT, то это одно изменение продукта в одном месте приведёт к переписыванию почти всех тестов. Если для СУБД такие пожелания пользователей – редкость, то для менее стандартных программ – обычное дело.**

Ууу, батенька, да если так пошло, то это просто диагноз. Давеча беседовал со своим куратором на работе, надо было foreign key-ю атрибутику поменять, чтобы шло удаление. Так вот задал я ему вопрос, как мне всё-таки это дело организовать? В миграции для БД? На что он мне взял и заявил резко, но правильно: нет и ещё раз нет, исключительно в логике программы. Почему? Да потому что начнут ребят использовать другой сервер, или, вот, пользователь захочет заниматься такой чушью, то днями и ночами придётся весь маппинг к БД переписывать и перестраивать. Кому это надо?!

Так и тут. Тесты должны проверять логику программы, а не формализмы разные, вроде словечек SELECT & SEARCH. Да, это понадобится, если мы хотим оптимизировать что-то, но это уже отдельный разговор, я б даже сказал, постскрам какой-то. А на производствах до этого пострскрама ещё доползти целым надо…

**Сбои, которые могут выловить тесты (повторяемый сбой в модуле, ранее уже исправлявшийся), – довольно редкое дело. Гораздо чаще сбои возникают в интеграции разных технологий, которые сложно автоматически протестировать. Например, при работе под таким-то браузером при таких-то настройках вот этот JavaScript работает неправильно.**

Редкое, однако, это зависит от платформы и развитости подходов к тестированию на ней. По поводу интеграций – согласен. Но всё это запросто можно отследить и протестировать, QA умеют.

**Наличие модульных тестов сильно затрудняет масштабный рефакторинг.**

Да оно даже микро рефакторинг усложняет.

**Одни из наших конкурентов широко используют agile-методы и TDD, но что-то оно им не очень помогает писать безошибочный код. Мы сравнивали количество найденных проблем в течение месяца-двух после major release, у нас показатели лучше в разы, если не на порядок. Частый выпуск версий просто не позволяет им довести код до ума и провоцирует исправление старых и серьёзных проблем методом написания «залепени».**

**Если мы выпускаем продукт раз в 2 недели и находится серьёзная ошибка в версии годичной давности, то нам её нужно будет исправить примерно в 40 ветках. Конечно, править 40 веток кода никто не будет, пользователям сообщат, что ошибка будет исправлена в следующей версии, многие пользователи перейти на неё не смогут к большой радости конкурентов с предложениями типа competitive upgrade offer.**

Верно.

**Излишне религиозная атмосфера превратила вполне здравую и работающую с 1970-х годов технологию модульных тестов в настоящий карго-культ. Адепты 100 % покрытия кода модульными тестами считают, что это обеспечит успех проекту.**

Верю, видел, знаю. Гиблое дело. Но тут такая вещь: балансировать в этом быстром мире да ещё и со скрамами и гибкими методологиями едва получается, кренит то в одну сторону, то в другую.

Понятия не имею почему, но сейчас, я представил себе it-сферу этакой мамашкой, у которой миллионы раскрытых ртов, которых надо кормить, причём кормёжка это отбор еды у других детёнышей. Так ладно, тут есть эти, вокруг ещё больше рождается, а мамашка не справляется и начинает тихонечко сходить с ума. А много детёнышей появляется, потому что о мамане много всего такого хорошего говорят, все вокруг её расхваливают. Ну и от мужиков-отраслей отпора нет. И получается, наверное, что от этого мамашка, которая была более-менее сконцентрирована в себе, в университетах и философско-математических концепциях, решила пошалить. Отчего и стала вести себя, как проститутка – прыгать на каждого и рекламировать себя не лучшей стороны, забыв о том, какой была изначально.

Печально это, ой печально. Как из этого оврага выезжать все вместе будем? Не знаю, наверное, только взрослеть, взрослеть…

**В завершение темы было бы непростительно не вспомнить о производственной системе Toyota, которую почему-то считают основой того же скрама. Ключевой особенностью системы в Тойоте является принцип «дзидока» (jidoka), означающий самостоятельность людей в управлении автоматизированной производственной линией. Если рабочий видит нарушение качества продукции или хода процесса, он имеет право, повернув соответствующий рубильник, остановить всю линию до установления причин дефектов и их устранения.**

**Внедрение таких методик вполне возможно и вне рамок японского общества. New United Motor Manufacturing Inc (NUMMI) – знаменитое совместное предприятие Toyota и General Motors, ещё в 1970-х годах вошло в «кейсы» бизнес-школ как пример повышения эффективности и качества через смену культуры работы.**

**Качество программного продукта – многозначное и сложное понятие. Производственная культура – ещё более сложное. В одном можно быть уверенным: ни о какой культуре софтостроения не может идти и речи, если любой программист из коллектива не способен остановить бессмысленный циклический процесс для выяснения, какого же рожна по историям заказчика потребовалось обобщать четырёхногих коров и обеденные столы.**

**Как вы заметили, я стараюсь перемежать темы, несколько напрягающие мыслительные органы, с занимательными зарисовками из жизни. Продолжим традицию историей о совместимости компонентов сложной тиражируемой системы.**

Заметили, это точно. Пару конспектов назад уж как заметили. Мне кажется, что гораздо полезнее для нас, интересующихся этой темой, было бы не читать разглагольствования автора и искать там суть, а просто с ним пообщаться и таким образом получить необходимый опыт. А то получается так: мы пишем конспект по материалу, который весьма вторичен, часть его отсырела, часть его вообще непонятна. То, что осталось, носит свою специфику, которую очень трудно обобщить, тем более нам, «зеленоватым студентам».

Посему, считаю, что смысла в частых заметках, взятых из этого материала, нет. Ограничимся мелкими. Но ёмкими.

**Итог главы:**

Какие же всё-таки Microsoft потешные. И не они одни такие, просто такого рода «фейлы» (fails) прочно ассоциируются именно с ними, с их продукцией. И, тем не менее, их продукция остаётся каким-то призрачным гарантом качества, благо, дизайн уже медленно подыхает.

А вообще, для меня мораль сей басни была такой: корпоративное решение другому корп. решению рознь. А мелкософты продолжают мутить воду, от которой страдают и разрботчики, и весь проект в целом (от простоя).

Возможно, что я не увидел чего-то ещё, что хотел вложить в рассказ автор. Что ж, жаль.

**Программная фабрика: дайте мне модель, и я сдвину Землю**

**В управляемой моделями разработке и в программной фабрике наиболее интересной возможностью является генерация кода, скомпилировав который, можно сразу получить работающее приложение или его компоненты. Мы проектируем и сразу получаем нечто работающее, пусть даже на уровне прототипа. Уточняя модели, мы на каждом шаге имеем возможность видеть изменения в системе. Проектирование становится живым процессом без отрыва от разработки.**

А вот это интересный и полезный подход. Во многом, он зависит от платформы, на которой выстраивается взаимодействие с базой, моделью и так далее. Если платформа очень перенасыщенная деталями, то от проектирования она будет отвлекать на хотфиксы и описанные ранее консультации с ребятками из microsoft. Иначе можно спроектировать не совсем гибкий по функционалу слепок будущей системы, которую можно уже переносить со всеми оговорками на более изощрённую платформу. К примеру, начать разработку на RubyOnRails и перенести всё на Java.

**В качестве решения перечисленных проблем появились так называемые двусторонние CASE-инструменты (two way tools), позволяющие редактировать как модель, непосредственно видя изменения в коде, так и, наоборот, менять код с полу– или полностью автоматической синхронизацией модели. Зачастую, такой инструмент был интегрирован прямо в среду разработки.**

Вот к таким вещам у меня двоякое отношение, тем более, зная раннюю позицию автора. Да, автоматизировать процесс разработки круто, как говорят, «базару нет», но, не стоит забывать, что смысл работы программиста не в том, чтобы обойтись нажатием 2-3 кнопок и получить готовый результат. В первую очередь, программисты, это люди, которые шевелят мозгами, ищут и исправляют ошибки, радуются написанию хорошего кода и постоянно совершенствуют свой стиль. О каком тогда совершенствовании может идти речь в таком гиперболизированном случае, когда автоматизирован весь процесс разработки? Единственное, это создавать автоматы. Сначала для программ / продуктов, а затем и для создания самих автоматов. Вот вам и рекурсия, круг замкнулся. Хорошо это или плохо? С моей точки зрения, скорее хорошо, ибо придётся напрягать мозги ради изобретения автоматов и совершенствования изобретённых, но всё равно, область применения ума может заметно сузиться, а это огроменный минус.

Надеюсь, такого никогда не произойдёт, хотя бы потому, что автоматизировать всё нельзя.

**Лень – двигатель прогресса, особенно когда надоедает переписывать генераторы кода и подстраивать относительно стандартные модели под частные требования.**

Ой, прямо в точку! Ради этого, собственно и нужна автоматизация. Кому нравится создавать кубики с полного нуля для того, чтобы построить небоскрёб? Чёрт возьми, да кто-нибудь занимаясь веб-разработкой начинает с реализации сетевых протоколов и описания структур в плюсах через код ассемблера? Знать, как устроены кубики необходимо, но очень вредно собирать то, что уже давно есть и широко используется, с абсолютного нуля на реальном производстве. ~~Жопа горит.~~ Сроки горят. Да, это необходимый этап, если ты создаёшь что-то инновационное или же учишься (вспомнить хотя бы УП). Но иначе это просто трата времени. Надо просто уметь разбираться в документации так, чтобы БЫСТРО найти необходимое и это применить.

**Лампа, полная джиннов**

**Метафора системы достаточно проста: хочешь генерировать код компонента или слоя – попроси об этом соответствующего «джинна» в форме стандартного «заклинания».**

Верно. А тут уж раздолье для всяких мнемонических приёмчиков, которые зависят от платформы и фреймворка.

**Язык создан на основе XML, поэтому делать описания можно непосредственно руками в обычном текстовом редакторе.**

XML – прошлый век. JSON лучше хотя бы потому, что его легче парсить в какой-нибудь словарь / хэш, который потом можно отдать на растерзание методу из ORM. А XML парсинг… я помню, мы его писал в рамках учёбы. Это была жесть. Плюс, XML затрудняет чтение, на мой взгляд.

**Приведу пример описания из рабочего проекта, содержащего один пользовательский тип, один перечисляемый тип, две сущности и одну связь (отношение) между ними.**

Это ужас. А где же компактность?! Может быть, я привык к YAML формату из Ruby On Rails, но XML это какое-то кощунство…

**Заключительная часть конфигурации представляет собой описания шаблонов.**

А вот это хорошо, если ориентироваться и знать. Это дополнительная гибкость, а это плюс.

**Слой хранения (СУБД)**

**Слой домена (NHibernate)**

**Слой веб-служб и интерфейсов доступа (ServiceStack)**

**Программа-клиент**

**Остановиться и оглянуться**

**Если рассмотреть метрики относительно небольшого проекта, то 40 прикладных сущностей в модели, состоящей примерно из 600 строк XML-описаний, порождают:**

**• около 3 тысяч строк SQL-скриптов для каждой из целевых СУБД;**

**• порядка 10 тысяч строк домена;**

**• 1200 строк XML для проекций классов на реляционные структуры (таблицы);**

**• около 17 тысяч строк веб-служб и интерфейсов.**

**Таким образом, соотношение числа строк мета-кода описания модели к коду его реализации на конкретных архитектурах и платформах составляет около 600 к 30 тысячам или 1 к 50.**

**Это означает, что оснащённый средствами автоматизации программист с навыками моделирования на этапе разработки рутинного и специфичного для платформ/архитектур кода производителен примерно так же, как и его 50 коллег, не владеющих технологией генерации кода по моделям. Любое внесение изменений в модель тут же приводит в соответствие все генерируемые слои системы, что ещё более увеличивает разрыв по сравнению с ручными модификациями. Наконец, для генерируемого кода не нужны тесты. Производительность возрастает ещё как минимум вдвое.**

**Даже если принять во внимание, что доля рутинного и прочего инфраструктурного кода по отношению к прикладному, то есть решающему собственно задачи конечных пользователей, снижается с масштабом системы, есть о чём поразмыслить в спокойной обстановке.**

Я уже писал ранее, что я думаю по поводу автоматизации. Хорошая штука, но не стоит ей увлекаться. Да, здорово иметь под рукой инструменты, которые позволяют быстро соорудить каркас обычного приложения (без всяких наворотов, чисто модели, контроллер, вьюхи – в зависимости от используемого паттерна своё наполнение) и работать с порождённым шаблоном, подстраивая его под себя. НО надо не ограничиваться одним шаблоном, смотреть шире и знать, то кроме MVC есть хотя бы MVVM и, соответственно, знать, как ручками, в случае высадки на местность, кишащую аборигенами, написать приложение с выбранным паттерном, чтобы спастись.

**Cherchez le bug, или Программирование по-французски**

**Хаос наступает внезапно**

**В жизни каждого мало-мальски сложного программного продукта есть стадия, когда система проходит некий порог увеличения сложности, за которым наступает состояние, которое я называю «самостоятельной жизнью». Это ещё далеко не полный хаос, но уже давно и далеко не порядок. Все попытки как-то организовать процесс разработки программ, всяческие методологии, применение парадигмы конвейера, стандарты и административные меры худо-бедно, но помогают оттянуть этот критический порог на некоторое время. В идеале – до того момента, когда развитие системы останавливается и она, побыв некоторое время в стабильном состоянии, потихоньку умирает.**

Согласен. Добавить нечего.

**Одна из проблем организации промышленного производства программного обеспечения состоит в отсутствии каких-либо формальных описаний деятельности программиста. Можно определить в технологической карте, как работает сварщик или каменщик, но как пишет программу программист зачастую не знает и он сам. До художника, конечно, далеко не все дотягивают, а вот с деятельностью рядового журналиста «независимой» газеты непосвящённому в софтостроение человеку сравнивать вполне можно. Этакий ядрёный сплав ремесла, некого богемного искусства, со вкраплениями науки, вперемешку с халтурой, шабашкой и постоянным авралом. Попытки же принудить программиста делать однотипные операции противоречат самой цели существования программного обеспечения как самого гибкого из существующих средств автоматизации рутинных процессов и потому изначально обречены на неудачу.**

**Кстати, если программист говорит вам, что в данном месте программа «должна работать», это значит, что с очень большой вероятностью она не работает, а он её просто не проверял в этом месте. Если же программист уже после обнаружения ошибки говорит: «А у меня она в этом месте работает…», лучше сразу его уволить, чтобы не мучился.**

**Что-то с памятью моей стало**

**На самом деле я очень не люблю, когда в комнате, где люди работают большее время дня в сидячем положении и относительной тишине, начинает кто-то мельтешить перед глазами и шуметь. Думаю, многие будут со мной солидарны. Отвлекает.**

**Три дня в IBM**

**Консультант, дядька лет пятидесяти, исправил нам эти параметры и пожелал дальнейших успехов. Кстати, у них там вся команда консультантов и меняеджеров состояла из мужчин предпенсионного возраста – факт отрадный, если вспомнить откровенную и циничную дискриминацию по возрасту при приёме на работу во многих российских фирмах.**

*Вот приятно это читать, ибо не понаслышке знаю, каково это. Мой отец, по профессии инженер-конструктор-радиотехник, около 50 своих лет тоже пытался устроиться в наш ПВТ, парк высоких технологий. И вы знаете, что ему сказали, не смотря на то, что за спиной у отца 5 патентов и сотни изобретений? «Вы нам не подходите по возрасту». И ладно он шёл бы обычным рядовым программистом, код клепать. Нет, это работа для таких зелёных, как я сам, студентов. Мой отец брал рамку выше – тим лид, менеджер, юнит-менеджер. Отец и на ASMе писал, и на PHP с C++, и веб-технологиями занимается даже сейчас, причём каждый день на этом зарабатывает. Но, увы, на все его заявки в ПВТ ответили циничным отказом. Из-за чего я не могу воспринимать ПВТ в нашей стране должным образом, по крайне мере, без взаимного цинизма.*

**Удалось, буквально ткнув пальцем в небо, выяснить причину ошибки в модуле. Ею оказался настолько искусно криво написанный цикл в одной из многочисленных функций, что не сразу была видна его нехорошая тенденция при определённых условиях превращаться в бесконечный.**

*Вспоминается Константин Антонович Зубович и его заветы «не использовать цикл for». ☺*

**Хорошо там, где нас нет**

**Если фирма с 50 % национального рынка электронных платежей вполне может работать без системы управления версиями исходного кода и дублированием таблиц в базе данных, то что же тогда говорить о стартапе…**

*Я нахожу это странным. Очень. Да, конечно, у стартапа главное это реализованная идея, но… как же формализованная чистота и прагматичность, прущая изо всех щелей? Как то это непривычно, что ли. И мне кажется, что скорее именно мелкие компашки в первую очередь работают с системами управления версиями и следят за дубликатами таблиц, ибо если у них там будут проблемы, то из мелкой компании они ни во что не превратятся.*

**О технических книгах**

**Дефрагментация мозгов**

**Современное софтостроение заслуженно забыто наукой. Ну а что вы хотите, если на вопрос «Почему нет науки на конференциях и в публикациях?» получаешь однозначный ответ «Никакая наука рынку не нужна».**

Но в то же время, плоды науки нужны всем. А направлять в науку никто не хочет. Наоборот, выгодно рынку делать людей бестолковыми.

**Нынешняя «гуглизация» позволяет быстро находить недостающие фрагментарные знания, зачастую забываемые уже на следующий день, если не час. Поэтому ценность книг как систематизированного источника информации, казалось бы, должна только возрастать.**

**Кроме стоящих вне конкуренции учебников остаётся в основном только конкретизация – узкоспециализированная проработанная технологическая тема ли-бо обобщение концептуальных наработок и практического опыта.**

**Долгий опыт университетского преподавателя Павла Андреевича Калугина ныне обогатился добавлением в курс информатики такой эклектики, как Enterprise Java. Проблемы подобные курсы вызывают больше не у студентов, а у обладающего системным образованием и мышлением преподавателя, вынужденного каким-то образом связывать противоречивое и выстраивать логику там, где её изначально не было.**

**Почему же относительно легко студентам? Дело, прежде всего, в смене образа мыслей, а может быть, и восприятия самой действительности. Мир – это такая очень большая и сложная компьютерная игра, а преподаватели, соответственно, должны обучать не премудростям стратегий познания мира, а практическим приёмам, секретным кодам и даже шулерству, чтобы пройти в этой игре на следующий уровень. Этакие «мастера ключей» из Матрицы. Про то, что в игре продукты на столе появляются прямо из холодильника, тоже стоит упомянуть.**

Да, за собой тоже такое замечаю. Но тут это самое шулерство во многом преподносится с красотой, отчего на душе становится благодатно и хорошо.

**Много копий сломано в обывательских дискуссиях о так называемом клиповом мышлении, шаблонности, «плохом образовании» вкупе с апокалиптическими прогнозами и прочим. Хотя на самом деле пока непонятно, к чему приведёт тенденция в перспективе ближайших десятилетий. Во фрагментарном «клиповом» мышлении есть и свои плюсы: способность быстро решать типовые задачи, широкая квалификация и мобильность, меньшие затраты на массовое образование. Минусы тоже ясны. В апокалипсис технократии я не верю, всегда будут рождаться способные дети и существовать ограниченное число учебных заведений, дающих жутко дорогое фундаментальное образование, вроде того, что было общедоступным в СССР. Видимо, этого должно хватить на поддержку критичных технологий цивилизации и дальнейшее их развитие.**

**Является ли фрагментарное мышление приспосабливанием к многократно увеличившемуся потоку информации? Отчасти да, только это скорее не адаптация, а инстинктивная защита. Чтобы осознанно фильтровать информацию о некоторой системе с минимальными рисками пропустить важные сведения, нужно иметь чётко сформированные представления о ней, её структуре и принципах функционирования. Если, например, у стажера нет знаний о СУБД в целом, то курс по SQL Server – конкретной её реализации, сводится к запоминанию типовых ситуаций и решений из набора сопутствующих практических работ. Вся теоретическая часть при этом просто не проходит фильтры.**

**В такой ситуации альтернативой ухода от информационных потоков и некачественного образования во фрагментарную реальность является самообразование на базе полезных книг. Потому что хорошая книга – самый эффективный способ дефрагментации ваших мозгов.**

Полностью поддерживаю. В особенности относительно самообразования. Считаю, что каждый должен им заниматься. В каждого это надо вложить. Иначе человечество просто погибнет от жажды наживы и развлечений.

**Простые правила чтения специальной литературы**

**Итак, настоящий исследователь должен:**

**• быть достаточно ленивым. Чтобы не делать лишнего, не ковыряться в мелочах;**

**• поменьше читать. Те, кто много читает, отвыкают самостоятельно мыслить;**

**• быть непоследовательным, чтобы, не упуская цели, интересоваться и замечать побочные эффекты.**

**Когда натыкаешься в Сети на очередное несдержанное восклицание «must have», подобная постановка вопроса коробит сама по себе. Представьте, что кто-то открыл для себя и полюбил варёный лук. И теперь если ты его не «must have», то как бы и не в струе, и вообще от жизни отстал. Смешно, конечно, но шутки шутками…**

А такое вообще везде, развелось много таких крикунов-максималистов.

**Если, пролистав с полсотни страниц, вы не сделаете ни одной пометки, то можете смело закрывать сие произведение. В топку бросать, наверное, не надо, но поберегите своё время. Если же к концу прочтения вам хватит листа А4, исписанного убористым почерком, для всех пометок, то можете считать, что чтение прошло не зря. Ну а если одного листа не хватило… Тогда смело идите на свой любимый интернет-форум и там поделитесь с коллегами своими находками. Когда вместо малоинформативного «must have», используя все тот же испещрённый пометками лист, вы сможете передать пользу от прочитанного, коллеги будут вам чрезвычайно признательны за рекомендации и собственное сэкономленное время.**

Здорово сказано.

На этом и закончим.