## Кто такой программист и немного о резюме (стр. 11-57)

### Конспект

#### Кто такой программист

**Программист минус алгоритмизация = кодировщик**

**Программист минус кодирование = постановщик задачи**

Практика многих десятилетий показала, что по-прежнему наиболее востребованным специалистом является инженер, способный как самостоятельно формализовать задачу, так и воспользоваться стандартными средствами её решения на ЭВМ. Вот его и следует называть *программистом*.

#### Иерархия подразделений



#### Иерархия должностей

«Б» – современные компании.





#### Уровни принятия проектных решений



#### Функциональные специализации (роли)

«Б» – современные компании.



#### Техническое творчество

Нельзя «сделать красиво», если относиться к работе исключительно утилитарно и шаблонно. Получаются сплошные типовые дома и «мыльные» сериалы. Необходимы и чувство прекрасного, и чувство меры, и знание других образцов, считающихся лучшими. Нужна техническая культура. Долгая работа, неблизкий путь, мотивация преодолеть который исходит, прежде всего, от любви к собственному делу, к профессии.

Но и нельзя «сделать красиво», если рассматривать софтостроение лишь как искусство и средство самовыражения. Любить себя в софтостроении, а не софтостроение в себе. Тогда красота рискует так и остаться не воплощёнными в жизнь эскизами. Невозможно обойтись без знаний технологий производства и хороших ремесленных навыков.

#### О квалификации

Отсюда неутешительный вывод для писавших программы в школьных кружках: количество проектов, где потребуется ваша квалификация, намного меньше количества некритичных заказов, а большинство ваших попыток проявить свои знания и умения столкнется с нелояльной конкуренцией со стороны вчерашних выпускников курсов профессиональной переориентации. На практике это означает, что вам, возможно, придётся снижать цену своего труда и готовиться к менее квалифицированной работе.

Не забывайте, что относительная доля критичных к качеству проектов падает, а переделка работающих систем базового уровня, от которых непосредственно зависит бизнес, – и вовсе редкое явление. Новые значимые проекты возникают только с новыми рынками и направлениями бизнеса.

Поэтому немало специалистов высокой квалификации уходят в экспертизу и консалтинг, где проводят аудит, обучение, «натаскивание» и эпизодически «вправляют мозги» разным группам разработчиков из числа переквалифицировавшихся.

Другой доступный вариант – специализация на предметных областях. В этом случае разработчик относительно автономен и, во-первых, гораздо менее ограничен в выборе инструментов. Во-вторых, что более существенно, доказывать кому-то степень владения инструментарием у него нет необходимости. К сожалению, хорошее знание предметных областей в сочетании с глубокими техническими знаниями платформ встречается редко в связи с плохой совместимостью высокоуровневых абстракций и низкоуровневых деталей. Обычная эволюция такого специалиста – системный аналитик, сохранивший знания технологий времён своего последнего сеанса кодирования в интегрированной среде.

#### Начинающим

Объявления вакансий:

1. «Быстро растущая компания» – фирма наконец получила заказ на нормальные деньги. Надо срочно нанять народ, чтобы попытаться вовремя сдать работу.
2. «Гибкие (*agile*) методики» – в конторе никто не разбирается в предметной области на системном уровне. Программистам придётся «гибко», с разворотами на 180 градусов, менять свой код по мере постепенного и страшного осознания того, какую, собственно, прикладную задачу они решают.
3. «Умение работать в команде» – в бригаде никто ни за что не отвечает, документация потеряна или отсутствует с самого начала. Чтобы понять, как выполнить свою задачу, требуются объяснения коллег, как интегрироваться с уже написанным ими кодом или поправить исходник, чтобы наконец прошла компиляции модуля, от которого зависит ваш код.
4. «Умение разбираться в чужом коде» – никто толком не знает, как это работает, поскольку написавший этот код сбежал, исчез или просто умер. «Умение работать в команде» не помогает, проектирование отсутствует, стандарты на кодирование, если они вообще есть, практически не выполняются. Документация датирована прошлым веком. Переписать код нельзя, потому что при наличии многих зависимостей в отсутствии системы функциональных тестов этот шаг мгновенно дестабилизирует систему.
5. «Гибкий график работы» – программировать придётся «отсюда и до обеда». А потом после обеда и до устранения всех блокирующих ошибок.
6. «Опыт работы с заказчиком» – заказчик точно не знает, чего хочет, а зачастую – неадекватен в общении. Но очень хочет заплатить по минимуму и по максимуму переложить риски на подрядчика.
7. «Отличное знание XYZ» – на собеседовании вам могут предложить тест по XYZ, где в куске спагетти-кода нужно найти ошибку или объяснить, что он делает. Это необходимо для проверки пункта 4. К собственно знанию XYZ-тест имеет очень далёкое отношение.

#### Про CV (резюме)

1. **Краткость – сестра таланта**. Даже в небольшой фирме ваше резюме будут просматривать несколько человек. Вполне возможно, что первым фильтром будет ассистент по кадрам, который не имеет технического образования и вообще с трудом окончил среднюю школу. Поэтому постарайтесь на первой странице поместить *всю* основную информацию: ФИО, координаты, возраст, семейное положение, мобильность, личный сайт или блог, описания своего профиля, цель соискания, основные технологии с оценкой степени владения (от «применял» до «эксперт»), образование, в том числе дополнительное, владение иностранными языками. Всё остальное поместите на 2–3 страницах.
2. **Кто ясно мыслит**, **тот ясно излагает**. Все формулировки должны быть ёмкими и краткими. Не пишите «узнавал у заказчика особенности некоторых бизнес-процессов в компании» или «разработал утилиту конвертации базы данных из старого в новый формат». Пишите «занимался постановкой задачи» или «обеспечил перенос данных в новую систему».
3. **Не фантазируйте**. Проверьте резюме на смысловые нестыковки. Если на первом листе значится «эксперт по C++», но при этом в опыте работы за последние 5 лет эта аббревиатура встречается один раз в трёх описаниях проектов, то необходимо скорректировать информацию.
4. **Тем более не врите**. Вряд ли кадровики будут звонить вашим предыдущим работодателям, но софтостроительный мир тесен, а чем выше квалификация и оплата труда, тем он теснее. Одного прокола будет достаточно для попадания в «чёрный список» компании, а затем через общение кадровиков и агентств по найму – ещё дальше.
5. Если соискание касается технического профиля, в каждом описании опыта работы **упирайте на технологии**, если управленческого – на периметр ответственности, если аналитического – на разнообразие опыта и широту кругозора.
6. **Не делайте ошибок**. Пользуйтесь хотя бы автоматической проверкой грамматики. Мало того, что ошибки производят негативное впечатление, они могут радикально изменить смысл фразы.
7. **Будьте готовы**, **что далее первой страницы ваше резюме читать не станут**, а о подробностях «творческого пути» попросят рассказать на первом собеседовании.

### С чем согласен

Шуточная манера написания книги. Много полезных советов (например, для резюме). Сравнение того как было раньше в сфере IT, и как это сейчас. Разбиение на функциональные роли.

### С чем не согласен

Опять-таки шуточная манера. Описание вакансий преподнесено так, как будто автора когда-то обидели. Очень много ненужного текста, который не имеет никого отношения к теме.

### Тема для эссе

Проведение собеседования и анализ резюме

## Железо и аппаратура (стр. 58-75)

### Конспект

Для аппаратуры используется модель конечного автомата. Она обеспечивает:

* полноту тестирования
* компонент работает с заданной тактовой частотой, то есть обеспечивает на выходе сигнал за определённый интервал времени
* внешних характеристик (состояний) у микросхемы примерно *два в степени количества «ножек»*, что на порядки меньше, чем у программных «кубиков»
* высокая степень стандартизации даёт возможность заменить компоненты одного производителя на другие, избежав сколько-нибудь значительных модификаций проекта.

Можно оценить количество состояний, которые необходимо охватить для полноты модульного тестирования при конвертации валют: ISO 4217 даёт список из 164 валют. Предположим, что наши входные данные:

* имеют только два знака после запятой;
* значения положительные;
* максимальная величина – 1 миллион;
* дата конвертации всегда текущая;
* мы используем только 10 валют из 164.

Итого: 102 × 100 000 000 = 10 000 000 000.

На практике же программист применит эвристику и будет тестировать, например, только несколько значений (один миллион, ноль, случайная

величина из диапазона) для нескольких типовых конвертаций из 100 возможных, дополнительно проверяя допустимую точность значений на входе. При этом формальный показатель покрытия модульными тестами по-прежнему будет 100 %.

Также необходимо программировать тест производительности, который тем не менее *не гарантирует* время отклика, а только позволяет определить его *ожидаемое значение* при некоторых условиях.

Таким образом, собрав из кучи микросхем устройство, мы уверены, что оно будет работать:

* согласно таблицам истинности;
* с заданной тактовой частотой.

### С чем согласен

Производительность «железа» возросла на порядки. Стоимость тоже на порядки, но снизилась. Увеличилась надёжность, развилась инфраструктура, особенно сетевая. Параллелизация вычислений пошла в массы на плечах многоядерных процессоров.

### С чем не согласен

С тем, что ООП сыграла не последнюю роль в вытеснении женского труда из отрасли IT (стр 73). По моему мнению объектно-ориентированный подход ну никак не мог этому поспособствовать.

Также не согласен с тем, что «ООП реализовано повсеместно без малейших представлений о его применимости». ООП активно используется в нынешнее время. ООП принесло в программы такой важный момент, как безопасность, возможность наследования, полиморфизма.

### Тема для эссе

Объектно-ориентированное программирование (его развитие, почему нужно использовать и что это дает)

## Web-приложения (стр. 76-95)

### С чем согласен

Согласен с написанным про апплеты. Они действительно были неудачной технологией. И также с написанным про создание web-приложений в прошлом, когда все писалось под конкретный браузер и что было непригодно после обновления браузера.

### С чем не согласен

Так как материал совсем не для дискуссии, а больше повествовательно-описательный, здесь трудно выделить что-то, с чем резко не согласен. Соответственно тяжело и выделить какие-нибудь важные темы для заметки.

### Тема для эссе

Написание кроссплатформенных web-приложений.

## Паттерны, ООП и БД (стр. 96-130)

### Конспект

#### Ад паттернов

Автор книги очень хорошо описывает проблему ООП в виде огромного количества паттернов, которые часто используются не там, где нужно. Вот какие проблемы выделяет автор:

* слепое и зачастую вынужденное следование шаблонным решениям;
* глубокие иерархии наследования реализации, интерфейсов и вложения при отсутствии даже не очень глубокого анализа предметной области;
* вынужденное использование все более сложных и многоуровневых конструкций в стиле «новый адаптер вызывает старый» по мере так называемого эволюционного развития системы;
* лоскутная интеграция существующих систем и создание поверх них новых слоёв API.

### С чем согласен

В принципе, все сказанное про ORM, по моему мнению, верно. ORM действительно упрощает жизнь разработчикам, но одновременно с этим, программисты не хотят разбираться в БД и языке SQL. Всегда нужно по правильному, сначала разобраться с БД, потом как с ней работать и что такое SQL, а только потом переходить к ORM, так как на деле будет допускаться огромное количество ошибок, некоторые из которых не вылезут сразу.

### С чем не согласен

Автор гнет очень жесткую линию относительно ООП. Он очень часто упоминает проблемы ООП. Но не нужно забывать, что каждая парадигма имеет свои плюсы и минусы. ООП является самым популярным подходом в настоящее время.

Также непонятна фраза о сложности распутывания кода в ООП. Мое личное мнение – это то, что разбирать чужой код в ООП намного проще, так как при ООП код довольно структурирован.

Агрегация и интерфейсы так же являются полезными вещами. Да, это все немного усложняет и увеличивает код. Но зато соблюдается структура и логика приложения.

### Тема для эссе

SQL, реляционные БД.

## Сервера и первая критика автора продукции Microsoft (стр. 130-169)

### Коспект

Определение с Wikipedia

Мейнфре́йм (также мэйнфрейм, от англ. mainframe) — большой универсальный высокопроизводительный отказоустойчивый сервер со значительными ресурсами ввода-вывода, большим объёмом оперативной и внешней памяти, предназначенный для использования в критически важных системах (англ. mission-critical) с интенсивной пакетной и оперативной транзакционной обработкой.

Основной разработчик мейнфреймов — корпорация IBM, самые известные мейнфреймы были ею выпущены в рамках продуктовых линеек System/360, 370, 390, zSeries. В разное время мейнфреймы производили Hitachi, Bull, Unisys, DEC, Honeywell, Burroughs, Siemens, Amdahl, Fujitsu, в странах СЭВ выпускались мейнфреймы ЕС ЭВМ.

### С чем согласен

«Как уберечь кукурузу от насекомых-вредителей? Очень просто: выкосить её всю, к чертям. Вредители придут, а кушать нечего.»

Так же согласен, что большие ЭВМ (типа мейнфреймов) являются неотъемлемой частью сервера. Никакие ПК не смогут заменить мейнфреймы!

### С чем не согласен

И снова критика, еще с первых страниц. Очевидно, что Windows 7 в 1000000 раз лучше, чем Windows NT. Увеличение требуемой ОЗУ – это техническое требование, так возможности ОС возросли в огромное число раз.

И снова критика x2. Чем автору не угодил Office 2007? В 7-м офисе был значительно расширен функционал. Автор явно любит использовать старые софт и технологии.

Автор обиженно в конце упомянул, что он перешел на Libre Office, в котором редактировалась книга. Так вот хочу сказать, что оформление книги ужасно. Много сухого текста, коды оформлены как обычный текст и т.д. Ну и хочется пожелать автору удачи с таким отношением к продуктам Microsoft. Если ему настолько все не нравится, то пусть переходит на ОС Linux (Unix, OS X, etc.) и пользуется самым “лучшим” и самым “удобным” софтом.

И как я и предположил, автор на 169 странице не забыл похвалить Linux, до этого сильно покритиковав продукцию Microsoft =) Не буду больше ничего писать, автор имеет право выражать свое предвзятое мнение.

### Тема для эссе

Почему OS Windows намного лучше OS Linux.

## Автоматизированная информационная система (стр. 170-201)

### С чем согласен

Очень хорошо расписано про слои АИС. Со всем, написанным про нее, согласен.

Согласен с написанным в конце про то, что бывает такое: много человек не могут придумать как работать с проектом, а маленькая команда 4-5 человек может как раз и придумать.

### С чем не согласен

Пункт про #ifdef. Зачем это вообще?

### Тема для эссе

—

## Управление предприятием, проектирование и сборка мусора (стр. 202-241)

### Конспект

В первой части очень подробно описывалась разработка продукта управления предприятием. Описание получилось довольно интересным.

Сборщик мусора значительно упрощает процесс разработки. Но все же в нем есть не только плюсы, но и минусы. Например, плохо, если никому ненужный объект будет висеть в памяти. Но зато он уменьшает сложность языка.

### С чем согласен

Проектированием системы должны заниматься не только технари. Все же должны участвовать люди, имеющие непосредственное отношение к сфере продукта (например, если это экономическая сфера, то проектировать должен заниматься человек, знающий ситуацию на рынке и в своей сфере)

Согласен с критикой книги Александреску «Мыслить шаблонно». Судя по написанному, там действительно много чего непонятного. Поэтому комментарии Тарасова очень даже удачны:

«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.»

«Контейнер всегда управляет своими объектами. Поэтому обращаться к его внутренним объектам нужно только через интерфейс самого контейнера.»

### С чем не согласен

Очень даже странно, но со всем написанным в этот раз соглашусь. Надеюсь, дальше темы будут интереснее

### Тема для эссе

—

## Транзакции и UML (стр. 242-259)

### С чем согласен

Со всем согласен. Автор подробно и интересно расписал про важные вещи, такие как синхронизация транзакций. Это действительно нужная вещь, т.к. покупатель будет недоволен, если ему продадут не так, как он заказывал, что негативно скажется на репутации магазина.

UML диаграммы очень часто помогают разобраться что и как должно реализовываться. К тому же сейчас есть много средств их построения.

### С чем не согласен

—

### Тема для эссе

Синхронизация транзакций

## Распределенный анализ данных и Оптисток (стр. 260-283)

### С чем согласен

Архитектура Оптисток выглядит довольно-таки интересно. Так как система является системой автоматизации продаж на выезде, то она является нужным и интересным продуктом. Как сказал сам автор, то архитектура типовая, но в его случае не обошлось без ограничений, а именно огромного числа пользователей. Но вместе с тем проблема была решена с помощью синхронизации и СУБД. Так же важен тот факт, что было уделено внимание GUI, так как интерфейс должен быть максимально понятным пользователю, чтобы упростить его работу.

### С чем не согласен

К сожалению, для меня темы, описанные на данных страницах, совсем не вызывают интереса. И мое мнение, что здесь не о чем рассуждать, так как это скорее истории из жизни автора.

### Тема для эссе

—

## Ревизия кода, модели жизненного цикла и тестирование (стр. 284-310)

### Конспект

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.

В любом софтостроительном процессе, будь то заказной проект или продукт для рынка, всегда можно выделить 4 основные стадии:

* анализ, чтобы понять «что делать»;
* проектирование, чтобы определить и запланировать «как делать»;
* разработка, чтобы собственно сделать;
* стабилизация, чтобы зафиксировать результат предыдущих этапов.

Ключевой особенностью гибкой методики является наличие владельца продукта (*product owner*), который лучше всех знает, что должно получиться в итоге.

Модульного тестирование, является краеугольным камнем всех гибких методик: если изначально неизвестно, что выстроится в итоге, дом или коровник, то подпорки у его стен должны быть в любом случае.

### С чем согласен

С необходимостью ревизии кода. Ревизия кода нужно не для слежения за выполнением проекта, но и для обучения молодых специалистов. Когда более опытный специалист производит ревизию кода молодого и указывает на ошибки, то это позволяет молодому специалисту поднять свой навык написания кода и предостеречь от появления ошибок.

Так же очень хорошо расписаны водопадная, итеративная и гибкие модели ЖЦ.

Владелец продукта — это просто иная формулировка старого правила «кто платит, тот и заказывает музыку».

Разработка модульных тестов – это тоже разработка. Для 100 % покрытия потребуется примерно столько же времени, сколько и на основную работу. А может, и больше, смотря как подойти к делу.

Практические выводы: соизмеряйте затраты на создание и поддержу автоматизированных модульных тестов с бюджетом проекта и располагаемым временем. Тем более плохо фанатично навязывать разработку «от тестов», умалчивая о названных особенностях и не учитывая другие возможности. Например, код, генерируемый по моделям, вообще не требует модульных тестов.

Автору понравился подход MSF. Это очень радует, так как в предыдущих главах он сильно критиковал практически всю продукцию Microsoft (за исключением старых продуктов).

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.

Качество программного продукта – многозначное и сложное понятие!

### С чем не согласен

—

### Тема для эссе

Подход MSF

## Система управления версиями и самогенерация кода (стр. 311-363)

### Конспект

**Team Foundation Server** (сокр. TFS) — продукт корпорации Microsoft, представляющий собой комплексное решение, объединяющее в себе систему управления версиями, сбор данных, построение отчётов, отслеживание статусов и изменений по проекту и предназначенное для совместной работы над проектами по разработке программного обеспечения.

**Subversion** (также известная как «SVN») — свободная централизованная система управления версиями, официально выпущенная в 2004 году. Цель проекта — заменитьсобой распространенную на тот момент систему Concurrent Versions System (CVS), которая ныне считается устаревшей. Subversion реализует все основные функции CVS и свободна от ряда недостатков последней.

**Лампа, полная джиннов**

Есть модель, описанная с помощью XML. Она проверяется, чтобы не было противоречивых фактов. Если модель корректна, «заклинатель» начинает призывать «джиннов» сделать свою работу, передавая каждому на вход кроме самой модели ещё и разнообразные параметры, конфигурацию, касающуюся не только самих джиннов, но и, например, таких настроек, как правила именования в конкретном слое системы. Обработав модель в соответствии с конфигурацией проекта, джинн выдаёт готовый к компиляции в среде разработки код. Для слоя хранения данных кроме генерации специфичных для СУБД SQL-скриптов производится их прогон на заданном сервере разработки. В случаях, когда система уже существует и подлежит, например, переделке, можно восстановить модель из схемы базы данных.

Слои:

* Слой хранения (СУБД) (по сути отвечает за конфигурацию БД)
* Слой домена (NHibernate) (этот слой реализует такую важную вещь, как маппинг (mapping) классов на структуры хранения
* Слой веб-служб и интерфейсов доступа (ServiceStack) (формирует интерфейс для работы (различные классы)

### С чем согласен

Довольно-таки поучительной получилась статья про TFS. По сути расписан один из возможных вариантов работы. Кратко его можно описать так: эту технологию можно, а ту нельзя; так, почему-то эта штука не настраивается, поищу-ка я в Google; О, вроде нашел, но оно все равно не работает:( ;…здесь такое понятие, как «танцы с бубном», т.е. попытки хоть как-то заставить работать нужную вещь; ну и завершается облегченным – «ну наконец-то!» Вот такая поучительная история:)

Также довольно интересно показалась идея про модель «Лампа, полная джиннов». Интересна тем, что она проста. И к тому же часто возникают мысли про что-то, что будет само генерировать код.

В принципе, дальше в книге описываются слои данной модели. Они являются вполне логичными, поэтому не имею ничего против. (слои написаны выше)

Эта модель так же завлекает тем, что, как утверждает автор, соотношение числа строк мета-кода описания модели к коду его реализации на конкретных архитектурах и платформах составляет около 1 к 50, что не может не радовать, так как писать нужно намного меньше кода.

### С чем не согласен

—

### Тема для эссе

Системы, которые генерируют код из какого-либо мета-кода.

## Работа с непрофессионалами, утечка памяти, IBM, размышления о софтостроении и литературе и подведение итогов (стр. 364-404)

### Конспект

Первые страниц 20 занимают рассказы из жизни автора. Он рассказывает, как "весело" он провел время в Париже и побывал целых три дня в IBM. Автор описывает, как он боролся там с насущными проблемами, а именно багами в коде, неопытностью или непрофессиональностью программистов и поджимающими сроками. Не буду вдаваться в подробности, но автор снова не забыл вспомнить о MS Windows (снова отрицательно; как сам сказал Тарасов: «в таком богатом и «**баг**атом» учреждении, как Microsoft»).

Довольно мило автор описал основателей компании (хотя зачем это?), но все же описание приведу здесь: «Основателей у фирмы двое: генеральная директриса Софи – экспрессивная француженка, **незамужняя**, уже в **летах**, и технический директор Блез – почти наш ровесник, но, по-видимому, реально взявшийся за свой первый проект». Так вот, автор описывает проблемы работы с двумя продуктами этой «конторы». В первом он столкнулся с неспособностью делать что-нибудь полезное тамошних программистов и с тем, что продукт съедал аж 1 ГБ в день. Поработав в воскресенье, он со своим **русским** товарищем смог исправить утечку памяти, но осталась проблема взаимной блокировки. Что с ней стало, дальше не указывается.

Работая над вторым проектом, автор успел побывать в IBM (ровестник Блез смог пробить путевку в известную нам компанию). Причиной было то, что у заказчика был сервер данной компании. Сначала его друг пытался справиться сам, как хороший знаток OS Linux, но потом на подмогу ему был отправлен автор. Опущу описание переездов, скажу лишь, что это очень долго и нудно.

Первые два дня автор и его друг не могли исправить проблему с каким-то модулем (он не запускался). Если кратко, то в первые два дня автор и его друг успели изрядно покопаться в разработках IBM (компиляторе, библиотеках и т.д.), покушать Snickers и изрядно наездиться. На третий день, же они сходу нашли проблему в коде модуля, и подключив некоторые библиотеки статически, они запустили наконец этот модуль.

И да, успешно сдали его Блезу, который отдал его довольному заказчику.

Последующие страницы заняты размышлением о науке в софтостроении, правильном подборе и чтении литературы и кратким изложением «Оснований». Здесь я везде целиком соглашусь с автором. Действительно, сейчас литература уже не так популярна. Все всё «гуглят». Очень показателен пример автора, где он рассказывает о том, как практиковал обучение СУБД. Старшее поколение пыталось вникнуть в детали основательно, а большинство из младшего — почти не обращало внимания, а лабораторные делали методом «тыка», выпрашивая код, чтобы сдать лабу. К сожалению, это очень правдиво. По учебе это видно, хватает таких личностей (но не все, конечно).

Правила чтения литературы просто приведу здесь. Там и так все хорошо написано:

Хороших и полезных книг в принципе не может быть много. Иначе вы посвятите все рабочее время чтению. Тут уместно вспомнить второе правило Аникеева [12]; кстати, все три его правила стоит привести целиком. 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.

Также понравилось следующее замечание, хотя скажу честно, сам нечасто делаю пометки. Может неинтересно мне эти темы... Так вот само замечание:

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.

Дальше автор сравнил литературу и софтостроение. А действительно, их организации очень похожи!



Заканчивает автор краткой повестью об том, как все появилось. Попытаюсь кратко изложить основные тезисы автора:

1. Сначала были слова, потом образовались последовательности => инструкции. Так появилось программирование и машинный код.
2. Появляются различные новые функции.
3. Ребята помоложе, придумали использовать вместо машинного кода мнемокоды. Так зародились трансляторы, так как ЭВМ не понимает мнемокодов.
4. И снова новые функции.
5. Тут пришла мысль о том, а почему не писать программы на человеческом языке? Так зародился транслятор, который переводил человеский язык в машинный код — Фортран (Формуло-Транслятор).
6. Появился КОБОЛ.
7. Пришла идея об типах переменных по умолчанию. И все переменные, начинающиеся на *i, j, k, l, m* и *n* Транслятор считал целочисленными.
8. Первые трансляторы требовали сложных инструкций. В целях упрощения появились Паскаль и С.
9. Ну а дальше пошли какие-то непонятные мне изменения: 4,5 система «Кофе» и т.д.

Закончу последним абзацем книги:

И тогда все хором – от Старых Чудаков Первой Системы до Самых Молодых Парней – сказали: в нашей отрасли кризис! А Старые Чудаки добавляют: застой, и света в конце тоннеля не видно. Рядом с ними Маргинальные Чудаки с искусственным интеллектом под мышкой, специализированными ЯВУ и ЭВМ, суперкомпьютерами, векторными вычислениями и прочей экзотикой стоят в сторонке от мейнстрима и посмеиваются над ловкостью, с которой им удалось обмануть всех, включая самих себя.

### С чем согласен

Не буду повторять выше описанное. Скажу в целом по книге. Я не сомневаюсь, автор — очень опытный и много чего повидавший человек. Он рассказывал обо многих насущных вещах в софтостроении и реальных попытках их исправить. Где-то удачно, где-то через «костыль», где-то неудачно, но главное, что попытки были. Так, конечно, книга была полезной. Но была бы еще полезнее, если бы в ней не было большого количества воды. Хотя это мое чисто субъективное мнение.

### С чем не согласен

Сколько можно критиковать Microsoft? Да, они не без изъянов. Но никто не идеален и ничто не идеально!!

Критика Microsoft мне очень не понравилась. Я помню, что автор верстал книгу в LibreOffice — бесплатный офис. Но верстка — ужасна. Хотя может в печатной версии все довольно-таки неплохо.

Так же не понравилось высказывание о том, что из-за ООП из программирования ушли женщины. Может автор что-то другое имел ввиду, но я понимаю так, как есть.

### Тема для эссе

—