Contents

1 Но зачем? (draft v2.1)

Эта заметка представляет собой текстовую версию моего выступления на мероприятии «Учиться учиться просто так не научиться» 14ого сентября 2013 года[1]. Целю мероприятия, как можно попытаться заключить из названия и существа моей персоны, являлось мотивирование собравшихся на осознанное [само-]образование в области математики и программирования. В частности, в своём выступлении я особенно сильно пытался убедить присутствующих, что функциональное программирование заслуживает больше внимания, чем ему традиционно принято уделять.

[1] видео-версия которого любезно произведена на свет Евгением Лукьянцом

Здесь приводится сильно отредактированный текст той беседы, тем не менее примерно соответствующий тому, что я говорил в реальности. Комментарии к этой текстовой версии беседы к приводятся *(в скобках курсивом)*.

Вообще, наверное, следовало бы разбить эту заметку на две, разделив общую образовательную мотивацию и радости Computer Science, но до тех пор пока у меня до этого не дошли руки самую свежую версию текущей реализации можно взять по адресу http://oxij.org/tmp/WhyLearnCS.pdf, а после её исчезновения из tmp следует искать в http://oxij.org/note/.

1.1 Вступление

Как можно было заключить из письма с объявлением *(которое тут не приводится)*, в процессе этого мероприятия я попытаюсь донести до вас ответ на самый главный вопрос жизни, учёбы и всего такого, формулировка которого, в отличие от аналогичного вопроса про вселенную, известна всем: «Но зачем?»

В этой беседе я попытаюсь ответить на следующие его вариации:

- «Я сюда пришёл. Но зачем?» — ответ на него больше всего интересует меня.
- «Учиться? Но зачем?» — ответ на него, вероятно, интересует вас.

После этого я объясню почему лично я люблю Computer Science и почему вам тоже столо бы начать, а в конце мероприятия предложу ответы на свой вопрос «Как?» и на ваши вопросы.

Да, если вдруг по дороге что-то оказывается непонятно в местах, специально не отмеченных как «места, в которых может быть непонятно», то нужно меня останавливать и спрашивать. Надеюсь, пока вопросов нет.

1.2 Мета-мотивация и идея применяемого метода убеждения

Люди по своей природе чрезвычайно дурные существа и занимаются серьёзным [само-]образованием всего по двум причинам:

- из-за наркотической зависимости к удовольствию от удовлетворения любопытства (aka «из внутренних побуждений»),
- из-за осознания непосредственной пользы получаемых знаний.

К сожалению, жизнь устроена так, что отсутствие достаточного уровня образования не позволяет ощущать непосредственную пользу от этого образования. В результате, многие люди изучают только то, что непосредственно прилегает к уже известным им областям и может быть сразу применено на практике, тем самым

ограничивая себя в накоплении критической массы знаний, необходимой для понимания непосредственной пользы от непосредственно *не*применимых знаний. В результате, знания у таких людей растут не примерно экспоненциально, как это должно быть в идеальном случае, а просто асимптотически приближаются к некоторому уровню, и если этот уровень меньше критической массы, то всё очень печально.

Указанный выше факт считаю столь важным, что отмечу его ещё раз: серьёзное отношении к учёбе со временем делает её (примерно экспоненциально) *проще и интереснее*, а её результат имеет *всё большее и большее* влияние на вашу жизнь.

Конечно, мне бы хотелось, чтобы вы просто поверили вышесказанному и начали серьёзно учиться, но люди чрезвычайно дурные существа, а

- учиться в сильном смысле этого слова объективно трудно *(по плану тут студенты второго и третьего курса должны начать кивать головой, а старшекурсники с сомнением покачивать)*,
- постоянно удерживать себя в наркотической зависимости от знаний во всевозможных областях, по видимому, невозможно (мне не удаётся, например),
- для ощущения пользы от накопления знаний нужно сначала набрать критическую массу этих самых знаний,

поэтому в этой беседе, после ещё нескольких мотивационных вступлений, я сначала буду пытаться убедить вас начать выбираться из потенциальной ямы беззнания пугая всевозможными гипотетическими отрицательными эффектами отсутствия образования (как общего, так и в математике, логике, computer science, программировании), а потом постараюсь заинтересовать вас приводя любопытные факты и примеры, про которые, как мне кажется, можно понять, что они имеют некоторую непосредственную пользу, даже не имея достаточного уровня образования (при этом польза от некоторых из них должна вас немного испугать силой потенциального эффекта на ваши дальнейшие перспективы в случае невладения ими).

1.3 «Верьте мне, ибо я стоял там, где были вы» или краткая история моей жизни

Начнём с того, почему мне кажется, что вам следует верить тому, что я буду говорить.

После поступления на КТ *(есть такая кафедра в университете ИТМО)* первые три курса я был весьма средним студентом — просто делал что мне говорили. Если просят делать что-то ближе к моим интересам — с большим интузиазмом, если нет

— то до тех пор пока не надоест. При наличии достаточного уровня терпения и постоянных мелких пинков различного рода особых проблем в процессе учёбы эта стратегия не вызывает, но, в целом, впечатления от такой учёбы как-то не очень. Знакомо?

«Первая космическая самоосознанность» у меня наступила очень поздно, собственно в конце третьего курса, когда я внезапно понял, что стоит начать учиться в сильном смысле этого слова. Мне просто повезло: до третьего курса я любил играться с разными операционными системами, а в начале третьего курса меня стало интересовать функциональное программирование и через него к концу года у меня таки набралась критическая масса знаний, заставившая меня начать интересоваться и всем остальным в технических областях.

Единственное, что меня печалит, так это то, что никто не «разбудил» меня раньше.

«Выход на вторую космическую самоосознанность» у меня наступил ближе к концу шестого курса, когда меня нагнала ценность всего того остального, в том числе философского и гуманитарного, что я в разной степени игнорировал до этого. Тут мне тоже просто повезло: на пятом курсе я решил доказать преподавателю философии, что в философии нет ничего сложного, и в результате начитал критическую массу материала в этой области.

Мораль этой краткой истории жизни в том, что, мне кажется, что я вполне представляю себе мотивацию (вернее, её отсутствие) такого «овощного» студента, коими в большинстве являетесь или вот-вот станете вы, потому что, в общем, я сам таким был.

1.4 Определения

Тут что-то может быть непонятно, но по ходу беседы всё должно проясниться. Пока просто задержите их в голове.

Образование Обучение + Воспитание. (постоянно повторяется А.А. Шалыто *(тут я рассказываю о том, кто это такой)*)

Programming Computer Science + Engineering. (в этой формуле я бы даже расставил коэффициенты)

Знание форма существования и систематизации результатов познавательной деятельности человека. (Википедия)

Умение освоенный субъектом способ выполнения действия, обеспечиваемый совокупностью приобретённых знаний и навыков. (Википедия)

Основная характеристика непробуждённого студента он заявляет «это не нужно!» при первой же трудности.

Разгильдяя умение притворяться, что ничего не делать — это нормально.

Приличный {**профессия**} {профессия}, обеспеченный приличным уровнем жизни при сохранении личных и профессиональных свобод.

{профессия} в последнем определении — свободная переменная. Например, «Приличный программист — программист, обеспеченный приличным уровнем жизни при сохранении личных и профессиональных свобод».

1.5 Карго культ

Начнём с лирического отступления про карго-культ [1]. *(тут я рассказываю про Карго-культ)*

При чём тут это?

1.5.1 Армия

Бегство от армии в ВУЗы — широко практикуемый в России карго-культ (это наша первая встреча с карго-культом «раз имею студень — значит студент», тут он аж на законодательном уровне). На КТ тех, для кого этот фактор является первоочередной мотивацией поступления в ВУЗ, вроде быть не должно.

1.5.2 Хороший ВУЗ

На мой взгляд, в любом хорошем учебном заведении должны выполняться два принципа:

- там учат люди, знающие в своей области несравнимо больше студентов (привет карго-культу «раз встал у доски — значит преподаватель»);
- студентов там заставляют учиться, а не разгильдяйничать (снова привет карго-культу «раз имею студень — значит студент»).

Без первого вообще не получается никакого обучения. Следствием второго фактора, в первую очередь, должно являться более низкое, чем обычно, содержание идиотов и разгильдяев в вашей проколотой окрестности, что должно мотивировать на ликвидацию своего беззнания

- из страха быть отчисленным,

- из наблюдения, что в вашей проколотой окрестности людям с бОльшим набором знаний многие вещи даются сильно проще (тут главное не начать говорить себе, что они «просто умные»[2]),
- а потом уже из-за ощущения приходящей пользы.

Вообще я считаю, что со временем подробность допросов на экзаменах должна только увеличиваться. В частности, можно разрешать студентам немного разгильдяйничать на первых курсах, давая им время «проснуться», чтобы случайно не отчислить нормальных, но «спящих», а с каждой сессией поднимать планку всё выше и выше. Также следовало бы разрешить перепроходить уже пройденные курсы и дать студентам почти свободно перетасовывать предметы в своём личном учебном плане.

К сожалению, на КТ же таких возможностей (сейчас?) нет, а после весенней сессии третьего курса все обычно расслабляются. Результат печален.

1.6 Наша мотивация

К мотивации. Поговорим о том, зачем нам преподавателям всё это надо. Как известно, денег за это платят, мягко говоря, не очень много.

Во-первых, учить кого-либо — хорошая мотивация, чтобы заткнуть все белые пятна в своей мысленной конструкции какой-либо области. По крайней мере на первое время в преподавании этой мотивации хватает.

Во-вторых, хотим мы этого или нет, вы — те с кем когда-нибудь нам придётся работать, или, что не менее вероятно, чьи ошибки придётся исправлять. Современное устройство общества, особенно не спрашивая вашего желания, выплюнуло вас из школы, дальше всякими механизмами отфильтровало желающих на эту кафедру, и будет кое-как содержать вас ближайшие шесть (или некоторых из вас соответственно разным причинам меньше) лет в обмен на то, что вы поддаётесь нашим попыткам сделать из вас приличных программистов. То, что много людей из различных софтверных компаний преподают на кафедре легко выводится из этого факта (кроме того, конечно, они ещё хотят забрать именно к себе самых вкусных из вас). Но и для тех из нас, кто занимается наукой, этот аспект тоже играет не последнее значение.

И, наконец, в-третьих, очень грустно слушать, когда люди рассуждают о том, в чём они явно ничего не понимают. Хочется это как-нибудь исправить.

[2] После чтения автобиографий и воспоминаний о нескольких великих учёных я окончательно убедился, что хорошо работающая голова вовсе не упрощает понимание сложных идей. Как говорил Джон фон Нейман «Young man, in mathematics you don't understand things. You just get used to them.» [2] «Просто умные» люди — это те, кому по разным причинам (семейным, социальным, биологическим, ещё каким-нибудь) к определённому возрасту повезло набраться знаний, нужных для того, чтобы в глазах окружающих считаться «умным». Хорошая голова может в этом только *помочь*.

1.7 Ваша мотивация, как она есть (вернее, как её нет)

Для борьбы с отсутствием мотивации к учёбе её следует разделить на два типа (всё как у философов — не знаешь с чего начать, начни с произвольной классификации):

- низкая общая мотивация («не хочется ничего делать»),
- отрицательная мотивация по конкретным предметам («не хочется учить то, что кажется не очень интересным»).

Две общечеловеческие причины немотивированности по обоим типам уже упоминалась в разделе выше:

- не интересно,
- не видно пользы.

В следующем разделе я сначала попытаюсь последовательно избавить вас от невидимости пользы общего типа, а затем разбавлю всё это аргументами в пользу изучения любимых мною областей знания.

1.8 Ваша мотивация, как она должна быть

У меня было два однокурсника (назовём их Андрей и Дима[3]) которые на старших курсах всё время вещали в эфир утверждения типа «на КТ никто ничего не знает, но все делают вид, что так и надо». Сначала я не обращал на это внимания, потом возражал на это замечаниями типа «в других местах ещё хуже», а теперь сам официально вам заявляю: «на КТ никто ничего не знает, но все делают вид, что так и должно быть».

Тут, однако, следует понимать, что в указанном утверждении главной частью является вторая. Первую часть не исправить, ваши (мои, чьи угодно) знания в любой момент времени состоят преимущественно из белых пятен. Но важно это осознавать, всё время печалиться по этому поводу, и всё время что-то с этим делать.

Теперь я попытаюсь вас убедить в том, что, этим стоит заниматься.

1.8.1 Утилитарная мотивация

Начнём с сугубо утилитарной мотивации: вам выгодно «проснуться» как можно раньше.

Очевидная иллюстрация. Чем плохо быть «абстрактным дворником», то есть иметь набор умений уровнем не выше необходимого дворнику? Даже при условии

[3] всё совпадения случайны

приличной заработной платы и престижности такой работы, вам будет очень тяжело найти другую работу, если условия перестанут вас устраивать или вас настигнет рефлексия и депрессия.

Аналогично, чем плохо быть «абстрактным программистом для/под 1С»? Переезд за границу превращает вас в дворника.

Чем плохо быть «абстрактным программистом для/под Web/Android/iOS»? Идею вы поняли, возражения пока придержите. Я специально выбрал программирование для Web и под смартфоны, потому что очень многие считают, что на этих технологиях какой-то свет каким-то клином сошёлся, а в том виде, в котором это существует сейчас, реальный набор необходимых для такой работы фундаментальных знаний бесконечно мал, и этот факт мы очень скоро обсудим.

Если вы думаете, что указанные риски не применимы именно к каким-то из вами любимых технологий, то вся история человечества играет против вас. Самый известный пример тех, кто слишком долго сидел и ничего нового не изучал и не придумывал, а потом оказался никому не нужен — луддиты «ломатели машин», но есть и недавние примеры, скажем, много ли сейчас нужно программистов для PalmOS, Windows CE, да даже просто под Win32?

Программирование для Web и под смартфоны тоже скоро очень сильно изменится и эффективное использование этих новых технологий будет требовать новых фундаментальных знаний, например, из области функционального программирования, современного проектирования операционных систем, криптографии и разного рода распределённых штук. Об этом мы ещё поговорим.

Теперь о возражениях, их к указанным причинам я нахожу два:

- можно очень хорошо зарабатывать на поддержке древних систем, владея древними технологиями (типа, например, Cobol);
- «когда какие-то новые технологии станут мейнстримом — тогда и буду их изучать (и наверняка новый мейнстрим в моей области всё равно не так уж сильно будет отличаться от старого), а сейчас мне и за это нормально платят».

Существует несколько причин не поддаваться таким аргументам.

В-нулевых, по поводу второго аргумента всё же вспомните про PalmOS и Win32. Перспектива, что вашу любимую технологию постигнет такая же участь не так уж и фантастична. К совершенно осязаемой нефантастичности такой участи мы ещё вернёмся ниже.

Далее, во-первых, как и для абстрактного дворника, если Cobol — это ваше всё, а завтра кто-то решит, что матожидание обслуживания поддерживаемой вами системы (в том числе, с учётом вероятности того, что вас завтра переедет автобус и нужно будет искать нового программиста на никому больше не нужном Cobol) слишком высоко, то вам придётся искать другую работу, а только с Cobol в кармане меня печалят ваши перспективы.

Во-вторых, этот аргумент касается обоих возражений и людей с намного более широким набором знаний и умений, как и для абстрактного дворника: если вас вдруг охватит рефлексия и депрессия по поводу своей работы, то, в отсутствие необходимых знаний, альтернативой переходу на такую же или ещё более грустную работу, вероятно, будет только суицид.

Для прикладывания последнего аргумента к программированию на Cobol нужно примерно ноль специальных знаний: копаться в древнем софте (aka «говно мамонтов») очень грустно, ибо древний говнокод с разы уродливее современного — там ещё и слабое железо заставляло всюду втыкать уродливешие хаки. Целенаправленная правка такого кода вызывает кровотечение из глаз.

Аналогично, если ваш уровень фундаментальных знаний по computer science достаточно высок, или даже скажем так, как только ваш уровень фундаментальных знаний по computer science становится достаточно высок[4], то почти всё современное программирование для Web начинает казаться вам очень грустным по одной простой причине: с точки зрения образованного программиста современное мейнстримовое Web-программирование представляет собой набор изощрённых способов конкатенации строк. Фундаментально, кроме семантик конкретных языков программирования (почти все из которых почти полностью совпадают), там нет вообще ничего. Программирование для смартфонов не на много веселее, большая часть приложений для них, в сущности, представляет собой нативные интерфейсы для Web-приложений, просто набор языков для программирования на них несколько более ограничен, зато есть доступ к каким-то локальным базам данных телефона и забавным (до тех пор пока не надоели) устройствами ввода («ой трясём-трясём мы наш смартфон»).

Рассуждая о последней причине некоторые люди приходят к выводу типа «меньше знаешь — крепче спишь», то есть, по их мнению, фундаментальные знания только мешают работать в мейнстриме — и это абсолютная правда, а потому они не нужны — а вот в этом я сейчас попытаюсь вас переубедить.

Таким образом мы приходим к третьей и самой главной причине — самостоятельности принятия решений. Она столь важна и общеприменима, что я выделил её в отдельный раздел.

1.8.2 Критическая мотивация

Слышали/видели ли вы когда-нибудь какой-нибудь «анализ» будущего какой-нибудь области IT или «обзор» возможностей какого-нибудь известного вам устройства работником какого-нибудь крупного средства массовой информации? Или даже так. Наблюдали ли вы когда-нибудь процесс «консультации» покупателей

[4] а поскольку для программирования для Web требуется примерно ноль фундаментальных знаний, то, при наличии мозга в голове, когда-нибудь он таки становится достаточно высок

в каком-нибудь компьютерном магазине? Давайте даже отбросим ошибки в рассуждениях, этим людям платят за убедительный анализ, а не за логически верный, просто обратите внимание на используемые ими предпосылки. Приведу несколько любимых мной технических примеров:

- чем больше частота процессора/шины/памяти — там быстрее работает машина,
- при прочих равных, картинка на мониторе с меньшим временем отклика будет лучше.

Рассуждать в подобных числовых критериях о софте значительно труднее, но это не страшно, всегда можно придумать какие-нибудь «словечки» (buzzwords) для создания «системы координат» свойств программного обеспечения, например:

- web 2.0, SaaS, PaaS,
- cloud,
- data mining, big data,
- паттерны проектирования

и тому подобное. В Википедии есть целая страница со списком [3].

Есть и более «жизненные» «факты», например, всеобщая истерия по поводу ГМО или рассуждения об отсутствии добавленной стоимости при покупке товаров через интернет.

Инженеры, системные и мультимедиа программисты слыша утверждения из первого списка обычно начинают хохотать (или плакать, после стольких-то лет). Большая часть словосочетаний из второго списка просто ничего не ознчает, а фундаментальные основы тех из них, что означают, существуют как минимум с 70-ых годов прошлого века, а некоторые даже раньше (иногда даже до появления компьютеров). И, да, я просто обожаю наклейки «без ГМО» на минеральной воде. Я вам гарантирую, что через несколько лет на ней станут писать «чистое ГМО», а на диетических продуктах — «не содержит фруктозы» вместо нынешнего «на фруктозе».

И при всём вышесказанном кто-то руководствуется этими критериями в принятии решений.

Хочется, чтобы вы осознали главное — если вы *чего-то* не знаете, то вам будут настойчиво пытаться промыть голову на эту тему, и, шансы таковы, что им это удастся. А потом на базе этой дезинформации вы будете принимать решения, от которых зависит судьба разрабатываемых вами программных продуктов и/или научных исследований. Наконец, если вы *о чём-то* не знаете, то кто-то другой принимает решение за вас.

Ни в серьёзной индустрии, ни большой в науке никому не нужны люди, которым всё время приходится говорить что надо делать.

Приведу несколько примеров про программирование.

- Любите программировать для Web? Знаете об Elm и Ur?
- Вы уверены, что парсинг XML в 3Мб/с, утыкающийся в процессор, во всех популярных веб-фреймфорках — это лучшее, что можно получить, прикладывая аналогичные усилия?
- Реализуете сетевой протокол? А доказать хоть что-нибудь о свой реализации сможете?
- Вы уверены, что Java — лучшее, на базе чего можно было построить операционную систему для смартфонов? Что вы знаете об Inferno OS?
- Почему вы считаете, что ваш новый проект нужно писать именно с использованием этих инструментов? Почему dotNet/JVM, а не что-то могущее в LLVM? Почему F#/Scala, а не Clojure? Почему тут у вас используется такой паттерн проектирования, а не просто написано как-то вот так? Что вы знаете о том, как аналогичный код выглядел бы на Haskell?
- Любите Sublime Text? А много ли вы знаете о том, как аналогичные вещи делают в Emacs? Как добиться этого в Vim, автоматизировать при помощи sed/awk?
- Используете системы контроля версий? А знаете, что можно делать при помощи git rebase? Пробовали ли использовать darcs?

Сразу несколько уважаемых computer scientistов постоянно отмечают, что современный программистский мейнстрим более подвержен моде, чем индустрия моды [2].

Ежедневно, косвенно или прямо, по глупости или специально, всем нам полоскают мозги, о каких-то из этих мифов даже смешно упоминать, о каких-то грустно, я знаю даже несколько таких, настойчивая попытка заявить о бредовости которых может привести вас в тюрьму (абсолютно серьёзно, и не только в России или ещё каких-нибудь странах «третьего мира», есть и работающие в развитых западных странах). Какие-то из этих мозгополоскательных аргументов разваливаются, если просто остановиться и немного подумать, но есть и такие, для понимания абсурдности которых требуются глубокие фундаментальные знания. Кстати, это то место, где я постоянно ощущаю пользу от философии, серьёзное изучение которой приучает к очень аккуратному изучению предпосылок и цепочки логических выводов в мыслях, изложенных на натуральном языке. Как писал Бертран Рассел в «Истории западной философии» [4] «изучая работы философа, нужно не пытаться опровергнуть написанное, а пытаться понять как так могло получиться, что написанное казалось ему верным» (мой вольный перевод).

Подводя итог этой мысли по отношению к программированию, я отнюдь не хочу сказать, что, скажем, в мейнстримовом программировании все принятые за вас решения очень плохи, но вы не можете оценить масштабы трагедии, а тем более начать придумывать способы выкручиваться в тех местах, где это можно сделать, если вы не знаете что вообще там можно такого сделать.

Теперь должно стать очевидным почему ранее я дал именно такое определение «приличного программиста». Уровень знаний приличного программиста позволяет ему с лёгкостью менять как минимум работодателей, а, в идеале, и предметную область (в некоторых рамках, конечно, сейчас у нас уже не древняя Греция). Да, работать с говнотехнологиями ему труднее, ибо он знает, что во многих местах можно было бы сделать значительно лучше и страдает по этому поводу, но, по крайней мере, он понимает где и как можно достичь хоть какого-нибудь компромисса между говнотехнологией и здравым смыслом.

1.8.3 Ландшафтная мотивация

Теперь я нарисую картинку, как-то иллюстрирующую мои представления об общем ландшафте фундаментальных знаний в окрестности Computer Science.

(тут я рисую картинку, которую можно увидеть в видео-версии беседы, в которую я сейчас дорисовал бы ещё несколько столбиков. TODO её сюда)

Суть этого изображения в том, что в общем ладшавште науки имеющей отношение к вычислениям на компьютерах чрезвычайно важную роль играет математическая логика и конструктивная математика. Например:

- Теория языков программирования (Programming Languages Theory, PLT) — это переодетая матлогика, языки программирования — это переодетые логические исчисления. Хорошим языкам программирования соответствуют хорошие логические исчисления и это наблюдение позволяет делать очень любопытные вещи (об этом мы ещё поговорим) и переносить результаты из области теории доказательств в область теории вычислимости и наоборот.
- Всякий вычмат — переодетый конструктивный матан. Каждому алгоритму вычислительной математики сооветствует доказательство какой-то теоремы в конструктивном матане.
- Физика занимается изучением поведения некоторых функций при помощи аппарата конструктивного матана, постоянно сверяя свои результаты с реальным миром.

К сожалению, в современной учебной программе на КТ (да и почти везде) этим фактам уделяется очень мало внимания.

Также на этой картинке следует нарисовать свой «фронт знаний» и оценить пропасть между ним, тем, чему вас научат к окончанию университета (если вы будете хорошо учиться), и что на самом деле следовало бы знать к этому времени.

1.8.4 Комбинаторная мотивация

Наблюдается следующий факт, который вы можете только принять на веру по крайней мере до тех пор, пока не «проснётесь»: скорость изучения в заданной и релейтед областях знания со временем возрастает экспоненциально. Скорость изучения в слабо релейтед областях меняется не так сильно, но со временем вырабатываются общие механизмы, позволяющие, как минимум, уменьшить «коэффициент трения».

Кроме того, я наблюдаю следующие факты.

- Многие мейнстримовые вещи представляют собой тривиальные комбинации фундаментальных фактов (это мы ещё обсудим).
- Фундаментальных фактов значительно меньше, чем их комбинаций.
- *Фундаментальные факты и идеи, после некоторой практики, быстрее всего изучаются в чистом или почти чистом виде.*

В результате, не «просыпаясь», вы со временем не только начинаете экспоненциально отставать в знаниях от лучших представителей своего поколения[5], со всеми вытекающими последствиями для ваших карьерных перспектив, но и тормозите своё изучение новых штук в горячо любимых вами мейнстримовых областях. В следующем разделе будем много примеров на эту тему.

1.8.5 Практическая мотивация

Теперь я покажу несколько любопытных вещей, которые должны убедить вас, что многие известные вам вещи скоро сильно изменяться.

Функциональное программирование *(тут я быстро рассказываю в чём суть функционального программирования от арифметических выражений до левой свёртки)*

Elm Теперь посмотрим на язык программирования/FRP-фреймворк Elm [5]. Обратите внимание на реализацию базы игры Марио в одну страницу кода [6].

FRP (Functional Reactive Programming) — это такой взгляд на программирование, когда всё, что делает ваша программа — это всего лишь левая свёртка внутреннего

[5] A wild exponential function appears. You use Differentiate. It is not very effective.

состояния по потоку входных воздействий при помощи заданной вами функции преобразования внутреннего состояния. Любую программу работающую с вводом-выводом на любом языке программирования можно представить в таком виде, просто функциональное программирование позволяет обобщить весь этот аппарат, написав его всего один раз для всех программ сразу. В результате этого обобщения получается FRP-фреймворк, например такой, как встроен в Elm, а от вас как от программиста требуется сконцентрироваться только на преобразовании внутреннего состояния программы.

Далее, следуя идеям функционального программирования, эту функцию-преобразователь внутреннего состояния представляют как набор маленьких функций, каждая из которых делает какое-то тривиальное преобразование (`jump`, `gravity`, `physics`, `walk`), а вся функция-преобразователь (`step`) — это просто их композиция.

В результате этого строго следования фундаментальным идеям получается Марио в страницу кода.

Ещё о пользе функционального Я утверждаю, что если вы хорошо знаете Haskell, то между C++03 и C++11 для вас нет почти ничего нового (кроме синтаксиса), а общие представления о Java позволят изучить добрый кусок языка Scala [2,7] за день. При этом в обратную сторону — изучить Haskell зная только C++11 или Scala — по моим наблюдениям, работает не так хорошо (но, конечно, лучше чем ничего). Всё потому, что Haskell очень принципиален в строгом следовании фундаментальным основам, по крайней мере, в общих для этих языков вещах. Изучая Scala вы узнаёте идеи, хорошо применимые в Java и Haskell (если учите серьёзно). Изучая Haskell вы узнаёте идеи, хорошо применимые в Scala, F#, Java, C, C++ (и не обязательно 11), C#, JavaScript, Python, Ruby и ещё куче языков.

Другой пример. Идеи, стоящие за языками Agda, Idris и Coq могут сильно помочь в изучении теории типов, математической логики, да и вообще математики. При этом Agda принципиально проще остальных, а после неё они осваиваются достаточно непринуждённо.

Если это всё ещё не убедило вас в нужности функционального программирования, а вы очень любите JavaScript, то советую погуглить «Promises Monad».

Если вы очень любите паттерны проектирования, то советую посмотреть [8] и [9].

Мораль этих ссылок в том, что

- хорошие библиотеки — это те, что строго следуют тем же фундаментальным принципам, которым следуют хорошие функциональные языки; нет, это не значит, что всё нужно срочно переписывать на Haskell, это значит, что нужно знать достаточно Haskell, чтобы писать хорошие программы на других языках,

- столь любимые мейнстреймом паттерны проектирования — тривиальные следствия этих фундаментальных принципов; если вы знаете Haskell и немного PLT, то паттерны проектирования не надо учить, они получаются сами собой при трансляции функционального кода в аналогичный объектно-ориентированный код.

Free Functions Но эта тривиальность мейнстрейма при должном образовании — не самое страшное, страшнее то, что следование упомянутым фундаментальным принципам позволяет компьютеру сочинять куски кода разной степени однообразности за вас. Добро пожаловать в мир автоматического доказательства теорем.

Оказывается, что в соответствии с изоморфизмом/соответствием Кэрри-Говарда [10] типам в программах на хороших языках программирования соответствуют логические утверждения. Например, `a -> b -> a` — тип функции (`const`) на языке Haskell, которая принимает нечто типа `a`, затем нечто типа `b`, а возвращает нечто типа `a`, говорит «из `a` следует, что из `b` следует `a`», или в символах, `a → (b → a)`.

Любопытно, что куча примитивов и управляющих структур в языках программирования имеют очень простые (не с точки зрения неподготовленного программиста, а с точки зрения теории языков программирования/математической логики) типы, а термы (программы) для этих типов можно автоматически «сочинять» без или почти без участия человека. Например, есть такая небольшая программа Djinn [11,12], которая выдаёт ответы на практические задания, задаваемые на экзаменах по языку Haskell, лучше среднего студента.

(тут я показываю как работает Djinn, а большая часть присутствующих печалиться тому, что страница кода на Haskell умнее их)

В случаях, когда требуется сочинить что-то более сложное, можно попросить машину сочинить искомый терм пользуясь не только нужным типом, но и набором тестов (уравнений), которым этот терм должен удовлетворять. Вообще я нахожу забавным, что в правильно типизированном языке использование Test Driven Development с правильно написанными тестами часто избавляет от необходимости написания собственно кода. Например, аналог dotNet LINQ на Haskell машина может сочинить автоматически из типов необходимых примитивов и нескольких уравнений.

Наконец, когда и этого не хватает, в дело вступают *тактики*. Вы как программист рассказываете машине о принципах написания нужной программы, а она, руководствуясь желаемыми типами и этими принципами, пишет эту программу за вас.

Да в этой области ещё не всё идеально, но прогресс идёт. На HaskellWiki есть страница во многом посвящённая этой теме [13]. Обо многих ли вещах оттуда вы

хотя бы слышали?

Тут следует задуматься о своих перспективах как программиста, если вдруг однажды (скоро) однообразное программирование, коим занимаются 95% современных программистов, будет автоматизировано.

(TODO в этот раздел понатыкать ссылок)

Операционные системы Я утверждаю, что поставив себе Arch Linux или Gentoo, а лучше оба по очереди, помучавшись с ними несколько месяцев, почитав manы и wiki, вы быстро разберётесь во всех идеях и подробностях организации работы UNIX-like систем и местного быта, чем очень сильно упростите себе дальнейшую жизнь. Всё потому, что как только вы хотите что-то автоматизировать или радикально подкрутить под себя, то кроме UNIX-like других вариантов у вас, в общем-то, нет.

PNaCl Теперь посмотрим на PNaCl [14], который когда-нибудь сотрёт разницу между обычным и web-программированием и позволит забыть обо многих современных инженерных ужасах в смежных областях (лишив работы ещё кучу необразованных программистов, выполняющих однообразный труд).

А ведь по сути PNaCl (да и NaCl) — это просто такой динамический загрузчик в стиле UNIX-like систем (почему-то[6] встроенный в браузер), проверяющий и изолирующий то, что он запускает.

NixOS Теперь посмотрите какая классная вещь NixOS [15]. Это такой дистрибутив операционной системы на базе GNU/Linux сильно отличающаяся внутренним устройством от привычных дистрибутивов. Результатом этого нестандартного устройства является, например, то, что

- *вся* система конфигурируется из одного файла с декларативным описанием того, чего мы хотим (а надстройки над NixOS позволяют конфигурировать целые кластеры машин из одного файла),
- поддерживаются атомарные апдейты и откаты (в том числе для кластеров, когда все сервисы на различных машинах перезапускаются в нужном порядке),
- аналогичным образом можно генерировать целые фермы виртуальных машин из одного файла конфигурации,
- поддерживается тестирование новых конфигурации в виртуальных машинах перед накатыванием на основную систему,
- поддерживается установка софта от обычных пользователей,

[6] потому что не все операционные системы столь же модульны, что UNIX-like системы

- можно держать одновременно установленными разные версии одного и того же софта,
- можно генерировать окружения с различными комбинациями различных версий софта,

ну и всё такое о чём вы могли только мечтать.
(тут я показываю репозиторий с конфигами своих машин)
Офигительно? Офигительно.

Базовая система в NixOS устроена совсем не так, как в других дистрибутивах, но UNIX-like система (GNU/Linux) выдержала такое издевательство над собой благодаря следованию некоторым фундаментальным принципам дизайна модульных операционных систем и чудесно работает. Теперь заметим, что NixOS построен на базе пакетного менеджера nix [16], в котором

- принципиально нельзя ошибиться с зависимостями пакета,
- результат сборки пакета строго детерминирован входными параметрами и зависимостями сборки.

А достигаются эти свойства благодаря тому, что nix представляет собой, кто бы мог подумать, специальный функциональный язык программирования.

Операционные система и логика *(тут я показываю какое отношение операционная система имеет к CPS-преобразованию и какая связь между ядром операционной системы и двойным отрицанием, об этом я когда-нибудь напишу отдельную заметку)*

Главная мысль Ещё раз, всё от того, что в этих примерах Haskell, Agda и UNIX-like — это системы очень близко следующие различным фундаментальным идеям, а потому опыт работы с ними легко переносится на другие технологии, использующие аналогичные идеи.

Если всё это вас не убедило, что надо изучать логику, функциональное программирование и операционные системы — то, я считаю, с вами что-то не так.

1.8.6 Do it yourself мотивация

Математика — самая офигительная из наук, для занятия ею не нужно ничего, кроме головы. Программирование — самая офигительная инженерная область: если вас что-то не устраивает, то всё, что не завязано на кривое железо и/или кривые

стандарты[7], всегда можно переделать с нуля, и для этого не нужно ничего кроме обычного компьютера.

1.9 Окей, ты меня убедил. Что надо делать?

1.9.1 Комикс о чувствах, испытываемых при образовании

Надо потыкать abstrusegoose #272. Мораль сего комикса, как её вижу я, в том, что вещи, которые принято считать очень простыми, должны выносить вам мозг. Если вам еженедельно что-нибудь не выносит мозг, то вы что-то делаете не так. Худшее ощущение, которое можно испытывать при образования — комфорт. Сильнее всего нужно набрасываться на незнакомые или даже пугающие вас области.

1.9.2 Что читать

(Когда-нибудь я сделаю отдельную заметку о том, что в каком порядке надо читать. Сейчас я просто перечислю несколько ключевых ссылок.)

- Aaronson. PHYS771 Quantum Computing Since Democritus
- [17]
- Susskind. Classical Mechanics

1.10 Как организовать workflow?

(Когда-нибудь я сделаю отдельную заметку о том как я борюсь со сложностью организации процесса. Пока тут будет просто несколько кейвордов и ссылок.)

- GNU/Linux (у меня NixOS, но это не обязательно, а в первое время даже вредно)
- git + git-annex
- Emacs + org-mode (+ evil). Самое главное тут — org-mode, смотри Org Mode - Organize Your Life In Plain Text!, мой воркфлоу — вариация на тему этой ссылки, но более научно-самообразовательно-ориентированный.
- xmonad

1.11 Ссылки

1. Культ карго. http://ru.wikipedia.org/wiki/Культ_карго.
2. TODO. Some link i could not find.

[7] при этом в первом случае можно поискать другую железку, а в обоих случаях можно попытаться сделать нормальный уровень абстракции и забыть об этих ужасах

3. List of buzzwords. http://en.wikipedia.org/wiki/List_of_buzzwords.
4. Russell B. A history of western philosophy.
5. Czaplicki E. Elm programming language. http://elm-lang.org/.
6. Super mario in elm. http://elm-lang.org/edit/examples/Intermediate/Mario.elm.
7. What are the differences and similarities of scala and haskell type systems? http://stackoverflow.com/questions/1815503/what-are-the-differences-and-similarities-of-scala-and-haskell-type-systems.
8. Akhmechet S. Functional programming for the rest of us. http://www.defmacro.org/ramblings/fp.html.
9. Does functional programming replace gOF design patterns. http://stackoverflow.com/questions/327955/does-functional-programming-replace-gof-design-patterns.
10. Curry-howard correspondence. http://en.wikipedia.org/wiki/Curry--Howard_correspondence.
11. Augustsson L. Hackage: Djinn. http://hackage.haskell.org/package/djinn.
12. Augustsson L. comp.lang.haskell.general: Djinn. http://permalink.gmane.org/gmane.comp.lang.haskell.general/12747.
13. Theorem provers. http://www.haskell.org/haskellwiki/Applications_and_libraries/Theorem_provers.
14. The Chromium Projects. Introduction to portable native client. http://www.chromium.org/nativeclient/pnacl/introduction-to-portable-native-client.
15. NixOS: GNU/linux distribution. http://nixos.org/nixos/.
16. Nix: purely functional package manager. http://nixos.org/nix/.
17. Sørensen M.H.B., Urzyczyn P. Lectures on the curry-howard isomorphism. 1998. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.17.7385.