ГЛАВА ВОСЬМАЯ

Опасности обучения на Java

29 декабря 2005 года, четверг

Ленивые детки.

Почему никто не хочет трудиться?

Верный признак начала старения: я начинаю ворчать и жаловаться, что «молодежь теперь не та» и что она не хочет или не может работать с полной отдачей.

Когда *я* был молодым, я учился программировать на перфокартах. Раз ошибешься – и карта испорчена, начинай все заново. У нас не было этих современных штучек, вроде клавиши Backspace, чтобы исправить ошибку.

С 1991 года на собеседованиях с программистами я обычно позволяю им выбрать для решения моих задачек любой язык программирования, который понравится. Тогда они в 99% случаев выбирали C.

Теперь они предпочитают писать на Java.

Поймите меня правильно: я вовсе не против использования Java в качестве языка реализации.

Пожалуй, стоит пояснить предыдущее высказывание. *В контексте настоящего обсуждения* я не собираюсь доказывать, что Java как язык реализации обладает недостатками. Эти недостатки многочисленны, но мы поговорим о них как-нибудь в другой раз.

Сейчас я имею в виду вот что: Java в целом недостаточно сложный язык программирования, чтобы позволить отличить замечательного програм-

миста от рядового. Этот язык может прекрасно подойти для каких-то работ, но речь сейчас не о том. Я бы даже сказал, что недостаточная сложность Java – это «фича, а не баг», тем не менее с этим языком связана отмеченная мной проблема.

Это дерзкое утверждение – результат моего наблюдения: две традиционные темы университетского курса программирования – указатели и рекурсия – многим учащимся оказываются не по зубам.

Раньше в колледже сначала читался курс по структурам данных – связанным спискам, хеш-таблицам и прочим типам, широко задействующим указатели. Такие курсы часто использовались для отсева: они были настолько сложны, что тот, кто с ними не справлялся, не мог рассчитывать на получение диплома по вычислительной науке, ибо если для вас слишком сложны указатели, то вряд ли вы одолеете теорию неподвижной точки!

Многие ребята из тех, кто в старших классах успешно писал для своего Apple II программы на BASIC, гоняющие по экрану мячик, записывались на курс «CompSci 101» (структуры данных), но когда дело доходило до указателей, их мозг перегревался, и вскоре они находили себе новую специальность вроде политологии, понимая, что юридический факультет – более удачный выбор. Я изучал статистику отсева среди студентов-кибернетиков – обычно это 40–70%. В университетах это считают потерями, но мне кажется, что отсеивать тех, кому программирование не доставит ни радости, ни успеха, просто необходимо.

Другим трудным начальным курсом для студентов-кибернетиков оказывался курс функционального программирования, в том числе рекурсивного. Очень высокий уровень этих курсов был установлен в MIT[1], где имелся обязательный курс (6.001) и учебник «Structure and Interpretation of Computer Programs»[2] (Abelson, Sussman) [The MIT Press, 1996]) – десятки, если не сотни лучших факультетов вычислительной техники использовали их как стандартное введение в специальность. (Вы можете – и должны – посмотреть старые варианты этих лекций в Сети.)

Сложность такого курса впечатляет. На первой же лекции вы узнаете достаточно много о Scheme и тут же изучаете функцию с неподвижной точкой, которая принимает на входе другую функцию. Посещая подобный курс CSE 121 в Пенне, я видел, что многим, если не большинству моих од-

---

[1] MIT – Massachusetts Institute of Technology (Массачусетский технологический институт). – *Прим. перев.*

[2] «Структура и интерпретация компьютерных программ».

нокурсников, он просто не по плечу. Я отправил преподавателю по электронной почте длинное письмо, пытаясь доказать несправедливость такого положения. Должно быть, кто-то в Пенне прислушался ко мне (или к другому жалобщику), потому что сейчас этот курс стали преподавать на Java.

Зря они это сделали.

Тут и зарыта собака. Многолетнее нытье ленивых студентов вроде меня и поступающие с предприятий жалобы на нехватку выпускников-программистов возымели действие, и в последние десять лет многие достойные в прочих отношениях учебные заведения почти повсеместно перешли на Java. Это модно, нравится кадровым агентам, которые оценивают резюме по методу «grep», но, главное, в Java нет сложностей, реально помогающих отсеивать программистов без той части мозга, которая распоряжается указателями и рекурсией, поэтому процент отсева снизился, на факультетах вычислительной техники стало больше студентов и увеличились бюджеты, так что довольны все.

Счастливчики, которых учат на Java, никогда не получат неожиданную ошибку сегментирования памяти, пытаясь реализовать хеш-таблицу, основанную на указателях. Им не придется ломать голову, пытаясь поразрядно упаковать данные. У них не встанут дыбом волосы от того, что в чисто функциональной программе, где значение переменной никогда не меняется, оно, тем не менее, постоянно меняется. Парадокс!

Они смогут получить высший балл по основной специальности и при отсутствии этой части мозга.

Может, я просто старый ворчун – из тех, кто гордится своим упорством, позволившим выбиться в люди несмотря на трудное детство?

Черт возьми, в начале прошлого века обязательными предметами в колледже были латынь и греческий – не потому, что от них было много пользы, а потому что считалось, что образованному человеку положено их знать. В некотором смысле мои аргументы аналогичны тем, которые приводились в пользу изучения латыни. «[Латынь] тренирует ум и память. Разгадывание латинских фраз – отличное упражнение для мышления, настоящий интеллектуальный пазл и хорошая тренировка логики», – пишет Скотт Баркер (Scott Barker, *www.promotelatin.org/whylatin.htm*). Но я не знаю ни одного университета, в котором сейчас была бы обязательна латынь. Может быть, указатели и рекурсия – это латынь и греческий вычислительной науки?

Я вполне готов признать, что 90% сегодняшнего кода обходится без указателей, а применять их в производственных программах просто опасно. Все правильно. И собственно функциональное программирование на практике используется не так уж часто. Согласен.

И все же владеть этими предметами необходимо в ряде самых интересных областей программирования. Например, не зная указателей, вы никогда не сможете работать с ядром Linux. Вы не поймете ни строчки кода Linux – да, фактически, любой операционной системы, – если не будете хорошо разбираться в указателях.

Тот, кто не понимает функционального программирования, не смог бы изобрести MapReduce – алгоритм, благодаря которому Google работает с гигантскими объемами данных, размещенных на многочисленных серверах. Термины «Map» и «Reduce» пришли из Lisp и функционального программирования. Понять, как работает MapReduce, может всякий, кто помнит из своего курса 6.001 или аналогичного ему, что у чисто функциональных программ нет побочных эффектов, поэтому их просто распараллеливать. Тот факт, что в Google смогли придумать MapReduce, а в Microsoft – нет, отчасти объясняет, почему Microsoft по-прежнему отстает в своих попытках наладить базовые функции поиска, тогда как Google уже решает новую задачу – построить Skynet^H^H^H^H^H^H, крупнейший в мире суперкомпьютер с с массовым параллелизмом. Я не уверен, что в Microsoft в достаточной мере понимают, как сильно они отстали в этом отношении.

А если копнуть чуть глубже, то действительная ценность указателей и рекурсии проявляется в том, что при создании крупных систем требуются гибкость ума, достигаемая в результате их изучения, и умственные способности, позволяющие избежать отсева после курсов, на которых они преподаются. Указатели и рекурсия требуют определенных способностей к рассуждению и абстрактному мышлению, а главное – умения рассматривать проблему на нескольких уровнях абстракции одновременно. Поэтому способность разобраться в указателях и рекурсии непосредственно связана с перспективой стать выдающимся программистом.

Преподавание вычислительных наук только на примере Java не позволяет отсеять студентов, у которых не хватает живости ума, необходимой для работы с данными понятиями. Как работодатель я вижу, что многие обладатели дипломов CS – выпускники заведений с полным обучением на Java – как программисты просто ни на что не способны, кроме как написать очередную бухгалтерскую программу на Java, хотя им и удалось проскочить через эту новую оглупленную систему курсов. Они ни за что не

справились бы с 6.001 в MIT или CS 323 в Йеле и, честно говоря, это одна из причин, по которым для меня как работодателя диплом CS из MIT или Йеля «весит» больше, чем диплом CS из Дьюка, где недавно полностью перешли на Java, или из Пенна, где Scheme и ML заменили на Java в том курсе, который когда-то чуть не доконал меня с друзьями, – CSE 121. Это не значит, что я не возьму к себе толковых ребят из Дьюка или Пенна, – возьму, но просто мне будет гораздо труднее оценить их реальные способности. Раньше я определял способных ребят по тому, как они за считанные секунды разбирались в рекурсивном алгоритме или, не задумываясь, писали функцию обработки связанных списков на основе указателей. Но столкнувшись с выпускником Java-колледжа, я не могу понять, чем вызваны его затруднения, – недостатком образования или отсутствием того участка мозга, который необходим для решения сложных задач программирования. Пол Грэм (Paul Graham, *www.paulgraham.com/avg.html*) называет их «дутыми программистами» (blub programmers).

Жаль, что Java-колледжи не отсеивают тех, из кого никогда не получится отличный программист, – хотя подобные заведения и оправдываются тем, что это не их проблема. Производство или, во всяком случае, кадровики, работающие методом «grep», в открытую требуют, чтобы студентов обучали Java.

Кроме того, Java-колледжи не развивают мозг своих студентов, и в результате им не достает ловкости, живости и гибкости ума, требующихся для того, чтобы хорошо проектировать программное обеспечение (я не имею в виду объектно-ориентированное «проектирование», в котором уйма времени уходит на бесконечное перетряхивание иерархии объектов или мучения по поводу надуманных проблем типа «как правильно – has-a или is-a?»). Нужно учиться мыслить одновременно на нескольких уровнях абстракции, и это именно тот тип мышления, который требуется для проектирования выдающихся программных архитектур.

Может ли изучение объектно-ориентированного программирования (ООП) стать адекватной заменой указателям и рекурсии в смысле отсева? Если ответить коротко, то нет. Не касаясь достоинств ООП, просто скажу – оно не настолько сложно, чтобы отсеивать посредственных программистов. Обучение ООП заключается, в основном, в заучивании ряда терминов вроде «инкапсуляции» или «наследования» и тестов с вариантами ответов, где нужно правильно выбрать между полиморфизмом и перегрузкой. Не труднее, чем запомнить несколько дат и имен в курсе истории. Проблема в ООП – это когда *ваша программа работает*, но ее трудно сопровож-

дать. Якобы. А проблема с указателями – это когда ваша программа выдает ошибку «Segmentation Fault», и у вас нет ни малейшего понятия о том, что случилось, пока вы не сделаете глубокий вдох и не заставите свой мозг работать на двух уровнях абстракции одновременно.

Я неспроста высмеиваю здесь кадровиков, действующих по методу «grep». Я еще не встречал такого программиста, который, зная Scheme, Haskell и указатели C, не смог бы за пару дней разобраться с Java и начать писать на нем программы лучше обладателей пятилетнего опыта работы с Java, – но попробуйте объяснить это типичному зануде из отдела кадров!

Какие же цели ставят перед собой факультеты вычислительных наук? Это же не профессионально-технические училища! Не их дело готовить рядовых работников для отрасли – для этого есть местные колледжи и государственные программы переподготовки для безработных. Университет должен давать студенту фундаментальные знания на всю жизнь, а не готовить его к тому, чтобы найти какую-то работу. Разве не так?

Итак. Вычислительная наука – это доказательства (рекурсия), алгоритмы (рекурсия), языки (лямбда-исчисление), операционные системы (указатели), компиляторы (лямбда-исчисление), откуда следует, что в Java-колледже, где не преподают C и Scheme, фактически не преподают вычислительную науку вообще. В реальном мире могут считать обсасывание понятия функции бесполезным занятием, но очевидно, что это совершенно необходимо для аспирантуры по вычислительной науке. Мне непонятно, как профессора вычислительной науки, составляя учебные планы, позволили программам своих учебных заведений опуститься до такого низкого уровня, когда выпускаются программисты только для производства, а не те, которые смогли бы окончить аспирантуру, получить докторскую степень и когда-нибудь занять их место. Хотя, погодите. Кажется, я понял.

На самом деле, если вспомнить дискуссии, которые велись в академических кругах во времена «Великого перехода на Java», то можно заметить: самым большим опасением было то, что Java – недостаточно простой язык для использования в процессе обучения.

Боже милостивый, подумалось мне, *да ведь они хотят еще больше упростить учебные планы!* Кормить студентов с ложечки! Пусть вдобавок ассистенты сдают зачеты вместо студентов, и тогда обучение в Америке перестанет кого-либо интересовать. Как можно чему-нибудь научиться, если учебный план тщательно составлен с целью до предела все облегчить? Кажется, есть даже специальная группа, которая должна выделить из Java достаточно простое подмножество, чтобы преподавать его учащимся,

и создать упрощенную документацию, скрывающую от неокрепших умов весь этот хлам EJB/J2EE, чтобы они не забивали свои головки разными курсами, которые им не понадобятся для решения все более простых вычислительных задач.

Самый мягкий ответ на вопрос, почему факультеты вычислительных наук так стремятся упростить свои курсы, заключается в предположении, что они надеются выкроить дополнительное время на теоретические понятия, если не придется целых две лекции объяснять студентам разницу между, скажем, Java int и Integer. Если это действительно так, то курс 6.001 – идеальное решение: Scheme – настолько простой учебный язык, что толковые студенты могут целиком освоить его за 10 минут, а оставшуюся часть семестра можно посвятить неподвижным точкам.

Фу.

Возвращаюсь к нулям и единицам.

(Вы изучали единицы? Повезло! У нас были только нули.)

ГЛАВА ДВАДЦАТЬ СЕДЬМАЯ

Бионический офис

25 ИЮЛЯ 2005 ГОДА, ПОНЕДЕЛЬНИК

Ну что ж.

Это заняло *гораздо* больше времени, чем ожидалось.

Наконец-то мы переехали в новый офис Fog Creek на 8-й Авеню, 535, спустя ровно десять месяцев после того, как я решил подыскать помещение взамен старого дома моей бабушки, где мы работали первые несколько лет, разместившись в спальнях и в саду.

Большинство менеджеров софтверных фирм знают, как выглядит хороший офис, как и то, что у них его нет и не будет. Похоже, проблема устройства офиса никогда не решается правильно, ничего не поделаешь. Компания арендует офис на десять лет, при этом менеджер команды разработчиков – последний, чьим мнением насчет дизайна помещения поинтересуются и кто увидит новые стойла... простите, офисные кабинки, только после переезда.

Черт возьми, уж в моей-то собственной компании я мог бы, черт возьми, сделать, как захочу! Вот и сделал.

Возможно, у меня завышенные требования к архитектуре. Вероятно, я чаще среднего программиста замечаю то, что меня окружает. Наверное, я слишком серьезно к этому отношусь. Но виной тому три причины:

- Есть масса свидетельств того, что правильная организация рабочего пространства увеличивает продуктивность программиста; особенно это касается личных кабинетов.

- Предлагая сногсшибательно роскошные личные кабинеты с окнами, гораздо легче нанять на работу суперзвезд, продуктивность которых в десять раз выше, чем у просто блестящих программистов. Если я в Нью-Йорке предлагаю зарплаты на уровне Бангалора, то мне нужны эти суперзвезды, поэтому на собеседовании хочу видеть их с отвалившейся челюстью. И это *драма*.
- В конце концов, это моя *работа*. Здесь я провожу *значительную часть* своей жизни вдали от родных и друзей. Так пусть здесь будет красиво.

В сотрудничестве с архитектором Роем Леоне (Roy Leone), при наличии большой площади (426 квадратных футов на человека) и прогрессивного CEO, я поставил перед собой цель создать совершенное рабочее пространство для разработчиков.

Вот какие «системные требования» (архитекторы называют это иначе) я дал Рою:

1. Личные кабинеты с закрывающимися дверьми абсолютно необходимы, это не обсуждается.
2. Программистам нужно много электрических розеток. Они должны находиться на уровне стола, чтобы можно было подключать разные устройства, не ползая по полу.
3. Необходимо, чтобы любые провода для передачи данных (телефон, сеть, кабельное телевидение, сигнализация и так далее) можно было легко перекладывать, не вскрывая стены.
4. Офис должен позволять программистам работать в паре.
5. Те, кто постоянно сидит за монитором, должны периодически давать отдых глазам, переводя взгляд на удаленные предметы, поэтому мониторы не должны располагаться вдоль стен.
6. Офис должен быть местом для встреч, где можно приятно провести время. Встречаясь с друзьями после работы, чтобы поужинать, вы должны *хотеть* встретиться в офисе. Как поведал Филипп Гринспен (Philip Greenspun) (*ccm.redhat.com/asj/managing-software-engineers/*): «Успех вашего бизнеса зависит от того, насколько ваш офис является для программистов средой обитания. Для этого нужно, чтобы офис был лучше дома среднего программиста. Тут есть два пути. Первый – нанимать программистов из трущоб. Второй – сделать офис красивым».

Рой прекрасно справился с работой. Вот за это мы и платим архитектору. Думаю, Рой может стать чем-то вроде международного эксперта по разработке офисов для программистов. Вот как он реализовал мои системные требования в трехмерном пространстве:

Личные кабинеты. Мы получили не только просторные личные кабинеты с окнами, но и общее помещение с рабочими местами для не-разработчиков, скрытыми в угловых альковах, так что каждый получил свое личное пространство, вне поля зрения других.

Стены между кабинетами и рабочими местами выполнены из высокотехнологичного полупрозрачного акрила, который мягко светится, пропуская естественный свет без ущерба для приватности.

Электронергия. Каждое рабочее место оборудовано двадцатью – именно двадцатью розетками. Четыре из них (оранжевые) подключены к источнику бесперебойного питания в серверной, так что UPS в каждом кабинете не требуется.

Розетки расположены чуть ниже стола в специальном коробе шесть на шесть дюймов, пролегающем вдоль всего стола. В нем можно аккуратно спрятать все кабели и закрыть его удобной крышкой под цвет стола.

Провода. Мы использовали систему Snake Tray, расположенную под потолком. Она начинается в серверном отсеке и проходит вдоль всего офиса, через каждый кабинет. Система полностью доступна, поэтому, если понадобится протянуть любой низковольтный кабель из точки А в точку Б, это делается легко и аккуратно. Мы переехали только в пятницу, и уже переделали офисную локальную сеть, что заняло у нас всего полчаса, так что кабель-канал Snake Tray оправдал возложенные на него надежды. В каждом кабинете есть свой 8-портовый коммутатор, к которому можно подключить свой ноутбук, *и* свой рабочий компьютер, *и* свой Макинтош, *и* тот старый комп, который вы держите у себя для того, чтобы читать «Joel On Software», когда основной компьютер перезагружается, устанавливая свежее обновление Windows, и у вас в запасе остается *еще* 3 порта. (Вниманию математических гениев: не нужно забрасывать меня письмами. Один порт служит для соединения с сервером.) Меня поражают глупые управляющие зданиями, которые все еще считают, что одного сетевого порта на кабинет достаточно. Может, это и так – для адвокатов.

Парное программирование. При установке стандартных Г-образных столов многие разработчики садятся в углу, поэтому, когда им нужно временно поработать с кем-то вместе, заняться программированием в паре или просто показать что-то на экране другому человеку, тому приходится

тянуться через весь стол или смотреть через плечо первого. Чтобы избежать этого, мы заказали длинные и прямые столы, поэтому, где бы ни сидел программист, всегда достаточно места, чтобы другой человек мог поставить свое кресло и сесть рядом.

Отдых для глаз. Хотя мы и установили столы вдоль стен, в стене есть внутреннее окно, в которое видно наружное окно *соседнего* кабинета. Благодаря столь замечательному расположению не нарушается уединение, поскольку, хотя ваше окно и выходит в соседний кабинет, оно направлено так, что из большинства точек кабинета вы видите только маленький краешек другого кабинета и его наружное окно. В результате в каждом кабинете – окна на три стороны, два из них смотрят наружу, что реализует архитектурную схему двухстороннего освещения в каждой комнате. Это большое достижение: попробуйте сами придумать такую планировку, чтобы в обычном здании у каждого работника получился угловой офис. Еще одна причина не жалеть о деньгах, потраченных на хорошего архитектора.

Отдых. Мы оборудовали в офисе небольшую кухню и гостиную с диванами, большой плазменной панелью HDTV и DVD-плеером. Мы также собираемся поставить там бильярд и игровую приставку. Личные кабинеты означают, что вы можете слушать музыку на умеренной громкости без наушников, никому при этом не мешая.

Считайте сами

Месячная аренда нашего офиса при полной занятости составляет примерно 700 долларов на одного работника. Перепланировка уложилась в смету и практически полностью оплачена домовладельцем. Подозреваю, что 700 долларов на человека ближе к верхнему пределу для программистов во всем мире, но если при этом мы можем нанимать людей из 99,9-процентили вместо 99-процентили, дело того стоит.