

# Два взгляда на программирование

Эдсгер Вибе Дейкстра

В окружающем нас мире мы можем встретить два радикально противоположных взгляда на программирование:

- Взгляд А: Программирование в основном весьма просто.
- Взгляд Б: Программирование — это очень сложно.

Это противоречие можно объяснить тем, что в этих двух взглядах одно и то же слово «программирование» употребляется в двух совершенно различных значениях, и на этом успокоиться. Тем не менее то, какой из взглядов преобладает, А или Б, оказывает глубокое влияние не только на кадровую политику организаций, использующих компьютеры, и учебные программы высших учебных заведений, но даже и на направление развития и исследований в самой компьютерной науке. Таким образом, представляется важным исследовать природу различий между этими двумя смыслами и по возможности выявить базовые предположения, которые делают каждый из них применимым. В этом и есть назначение данного документа.

В этом исследовании у меня есть одно препятствие: в этой дискуссии я не являюсь

нейтральной стороной. Я — убеждённый сторонник взгляда Б и рассматриваю взгляд А как основную причину многих печальных заблуждений. С другой стороны, я не считаю, что наличие собственного мнения дисквалифицирует меня как автора, особенно если я заранее предупрежу об этом своих читателей и не буду притворяться нейтральным. В процессе анализа мы раскроем, как эти различные взгляды на программирование (которое является человеческой деятельностью!) связаны с различными мнениями Человека. Это уже само по себе является весьма ценным пониманием, так как объясняет почти религиозное рвение, с которым разворачиваются сражения между сторонниками противоположных взглядов (догм?).

Ранняя история автоматических вычислений делает взгляд А очень понятным. До то-

го, как у нас появились компьютеры, программирование вообще не являлось проблемой. Затем появились первые машины: по сравнению с нашими нынешними машинами они были просто игрушками, и по сравнению с тем, что мы пытаемся делать сейчас, они использовались лишь для «микроприложений». Если на этом этапе программирование и было проблемой, то весьма незначительной. Добавьте к этому источники трудностей, которые в то время поглощали — или лучше сказать узурпировали? — большую часть нашего внимания:

1. Арифметические устройства были слишком медленные по отношению к тому, что мы хотели делать с их помощью: эти башмаки почти всегда оказывались слишком тесными, и ради

эффективности программы допускались все возможные трюки кодирования (и очень немногие из них реально не использовались).

2. Разработка и конструирование арифметических устройств были настолько новой и, следовательно, трудной задачей, что, если очередная аномалия в коде инструкции могла избавить от каких-либо кульбитов, от них обычно избавлялись, — также, разумеется, потому что мы имели так мало опыта в программировании, что мы не могли так уж хорошо распознавать «аномалии в программном коде»; в результате, помимо необходимости использования трюков в коде, также была великолепная возможность их применять.

3. Памяти всегда было слишком мало, и это вместе с ненадёжностью первого оборудования препятствовало более разумному использованию машин.

В это время программирование представлялось в первую очередь как битва с ограничениями машины, битва, которую нужно было выиграть хитростью. Это было систематическое использование специфических особенностей каждой машины: это был расцвет виртуозного кодирования.

В течение следующих десяти-пятнадцати лет процессоры стали в тысячи раз быстрее, памяти стало в тысячи раз больше, и языки программирования высокого уровня вошли в обиход. И именно в это время с одной стороны программирование всё ещё прочно ассоциировалось с тесными башмаками, в то вре-

мя как с другой — чувствовалось, что башмаки жмут всё меньше и меньше, и ожидалось, что ещё через пять лет технического прогресса проблемы программирования вообще исчезнут. Именно в этот период появился взгляд А. Именно в конце этого периода, вдохновлённый взглядом А, был разработан COBOL с провозглашённым намерением, что он должен сделать программирование, выполняемое профессиональными программистами, ненужным, позволив «пользователю» (не в это ли время слово «пользователь» стало общеупотребимым?) записывать то, что он хочет, на «простом английском», который любой может прочесть и понять.

Все мы знаем, что эта прекрасная мечта не воплотилась в жизнь. Следующие пять лет принесли нам вместо исчезновения всех проблем, связанных с программированием,

кризис программного обеспечения, и COBOL, вместо того чтобы выжить профессиональных программистов, стал грандиозным механизмом программирования для ещё большего их числа; и ещё десять лет спустя мы всё ещё имеем машины, в которых ошибки базового программного обеспечения вызывают в среднем час простоя на каждые пятнадцать часов работы. Очевидно, что до сих пор остались серьёзные проблемы программирования...

Забавная вещь: несмотря на полную очевидность обратного, взгляд А всё ещё жив. В объяснение этого странного факта некоторые с укоризной указывают пальцем на большие организации: либо на организации, применяющие компьютеры, которые, привлекая большие трудовые ресурсы, разделяющие взгляд А, тем самым потеряли возможность свободно расстаться с ним, либо на фирмы-



производители компьютеров и образовательные институты, которые поддерживают широко распространённый взгляд А, который им полагается представлять как основной для их рынка. Даже если этот палец грозит поделом, тем не менее я не могу принять это как полное объяснение живучести взгляда А, и вынужден предположить, что взгляд А удовлетворяет более глубокие, физиологические потребности.

Откуда взялся взгляд Б? Были люди, которые чувствовали, что появление больших и быстрых машин заменит тесные башмаки хотя бы на башмаки по размеру, и что несмотря на это эффективность выполнения программ останется серьёзной заботой программиста, заботой, которая станет даже более важной по мере роста машин и приложений, и что более сложные установки поставят бо-

лее трудные проблемы. Также было замечено, что переключение с машинного кода на языки высокого уровня вовсе не гарантирует тех преимуществ, на которые возлагали столько надежд. В частности, программисты продолжали столь же охотно выдавать большие куски непонятного кода, и единственное различие было в том, что теперь они делали это в более грандиозных масштабах, и что высокоуровневые ошибки пришли на смену низкоуровневым. Они также поняли, что появление языков программирования высокого уровня не уменьшает потребности в тщательности: избыточность языков высокого уровня лишь уменьшает вредный эффект от некоторых видов небрежности. И тогда появился взгляд Б. (Взгляд Б не является реакцией на кризис программного обеспечения, который всплыл в 1968, так как он на много

лет старше. Фактически взгляд Б предсказал этот кризис, но даже это подтверждение не убило взгляд А).

После этой интерлюдии по поводу появления взгляда Б вернёмся к нашему вопросу: как и почему, лицом к лицу с признанными проблемами программного обеспечения, взгляд А (а именно — программирование в основном весьма просто) всё ещё здравствует. Вот ответ: из-за веры, причём не веры в лучших программистов, а веры в лучшие языки программирования или (диалоговые?) системы программирования, а также веры в лучшие технологии программного менеджмента.

Я придерживаюсь мнения, что программирование — один из наиболее сложных разделов прикладной математики, поскольку оно также является одним из наиболее сложных направлений инженерии, и наобо-

рот. Когда я попытался разъяснить одному из моих коллег-математиков, почему я придерживаюсь этого мнения, он довольно бесцеремонно отказался выслушать мои доводы и вместо этого обвинил меня и моих единомышленников-компьютерщиков в том, что мы до сих пор не создали язык программирования, который сделал бы программирование настолько простым, насколько ему и подобает быть! Возможно, мне стоило бы спросить его, почему математики до сих пор не разработали нотацию, которая позволила бы любому, невзирая на отсутствие профессиональной подготовки, заниматься математикой?

Копнув чуть глубже, выясняется, что сторонники взгляда А не отрицают потенциальной сложности программ и их разработки, но верят, что жизнь программистов бу-

дет становиться всё легче, поскольку все наиболее сложные части задачи будет брать на себя машина. Они указывают на появление языков программирования высокого уровня, которые уже сделали программирование гораздо легче, чем во времена старых машин, и опрометчиво экстраполируют, что в будущем программирование станет вовсе тривиальным. Но оправдана ли эта экстраполяция? Я много программировал, как в машинных кодах, так и на языках высокого уровня, и последние несомненно более удобны, поскольку в этом случае многие решения, относящиеся к внутренним деталям программы, такие, как распределение памяти, не приходится принимать явно, поскольку ими занимается алгоритм распределения памяти компилятора. Переход к языкам высокого уровня освобождает нас от многих тривиальных

забот. Это сделало программирование деятельностью с небольшим количеством нудной работы, с преобладанием изобретательности: именно та часть работы, которая занимала целые дни, исчезла! Вывод, который следует из появления языков программирования высокого уровня, о потребности в программистах большего интеллектуального калибра, полностью подтвердился моими наблюдениями в Западной Европе (где я мог следить за разработками последнее время): в конце 60-х многие крупные организации, использующие компьютеры, испытывали проблемы в подборе подходящей работы для программистов, нанятых в 50-е годы, поскольку профессия переросла их интеллектуальные способности.

Однако ни эти наблюдения, ни указания на провал попытки COBOL'а выжить про-

фессиональных программистов не произвели никакого впечатления на правоверных. Они объясняют, что традиционные языки программирования высокого уровня потерпели крах из-за их «процедурности», и что провал COBOL'а очевиден, поскольку ввиду недостаточной интерактивности он на самом деле не является простым английским, но вот через пять или десять лет дальнейший прогресс в области Искусственного Интеллекта (ИИ) позволит нам построить «контекстно-зависимые», «основанные на знаниях», «автоматизированные системы для мышления и понимания», такие, что «пользователю достаточно будет только побеседовать с ними».

Должно быть, я неизлечимый скептик, но мне весьма трудно поверить, что подобным надеждам суждено сбыться. Имеются некоторые видимые подтверждения таких ожида-

ний — я цитирую отрывок из недавно полученного письма: «... в общем, передача более совершенному компьютеру того, что мы сегодня называем человеческими навыками, знанием и разумом». Я не намерен повторять здесь фрагменты горячих дискуссий, которые мы проводили по поводу значимости Искусственного Интеллекта, да здесь это и ни к чему.

Во-первых, оглядываясь назад, приходим к неизбежному заключению: смешивание надежды на будущее ИИ с завтрашней реальностью было бы безрассудством, и весьма безответственно быть неподготовленными на случай, если мечты по поводу ИИ останутся мечтами по крайней мере на протяжении нашей жизни. Или, говоря другими словами, глядя на серьёзность сегодняшних проблем программирования, обычная осторож-



ность заставляет нас не забывать о взгляде Б.

Во-вторых, первоочередной задачей программиста, если он хочет, чтобы его творения заслуживали доверия, будет разрабатывать их настолько понятными, что он сможет нести за них ответственность, и, несмотря на ответ на вопрос, как много из его нынешней деятельности может быть переложено на машину, мы должны всегда помнить, что ни «понимание», ни «ответственность» не могут быть классифицированы как деятельность: это скорее «состояние разума» и в принципе не может быть передано машине.

Я считаю неразумным, в особенности для учёного-компьютерщика, недооценивать влияние психологической школы, которая, считая человеческий разум слишком сложным и трудно поддающимся изучению, занялись

вместо этого изучением крыс, и даже ограничивает это изучение — как я слышал недавнюю формулировку — «наиболее механической формой поведения — зачастую настолько механической, что даже крысам не дают проявить свои высшие возможности». Представляя свои грубые, механические модели в качестве допустимого приближения к человеческому разуму, они опасно затуманили различие между человеком и машиной, и мы наблюдаем два взаимно дополняющих друг друга феномена: антропоморфный взгляд на машины и механический взгляд на людей.

Эта неразбериха, вне всякого сомнения, — плод усилий верховных жрецов ИИ. Преобладание в основном антропоморфной терминологии в компьютерной науке — «память», «интерпретатор», «язык программирования», «рукопожатие», «диалог», и это лишь немно-

гие примеры, — это предупреждение, которое не следует игнорировать. Я не знаю, как думать и разговаривать, обходясь без метафор; я знаю также, что каждая метафора несёт в себе опасность скрытого подтекста. В данном случае антропоморфной терминологии в компьютерной науке мы давно уже достигли стадии, когда опасность путаницы перевешивает достоинства аналогии.

Кроме того, кажется, что механический подход к людям среди компьютерных учёных (и их руководителей) распространён шире, чем представляется мне нормальным. Ибо я подозреваю, что именно этот механический подход ограничивает деятельность программистов механическими действиями по написанию кода, и затем измеряет «производительность труда программиста» количеством произведённых им строк кода. (Когда весь-

ма широко известный и очень уважаемый учёный-компьютерщик использовал недавно эту меру производительности труда программиста в своей лекции, от слушателей поступило предложение говорить не о «количестве произведённых строк кода», а о «количестве израсходованных строк кода», и что лектор, следовательно, занёс их не в ту графу учёта баланса расходов и доходов. Лектор ответил, что он вынужден использовать эту меру производительности, поскольку не располагает никакой альтернативной, которая позволяет вести точный учёт!) Это не может больше рассматриваться как безобидное заблуждение, поскольку принятие этой бессмысленной «меры производительности» профессиональными программистами гарантированно стимулирует их к написанию рыхлого кода.

Влияние психологии было рассмотрено

здесь, поскольку оно объясняет цепкость, с которой так много людей склонны тяготеть ко взгляду А.

В основном это не вина производителей компьютеров, которые желают вести дела так, будто они продают простейшую продукцию; и не вина руководителей программных проектов, которые предпочитают рассматривать деятельность программистов как простой и предсказуемый процесс; и не вина учебных заведений, которые хотели бы подготовить студентов к достижению гарантированного успеха.

Это — следствие комфортной иллюзии, что Человек — это лишь сложный автомат, которая, подобно наркотику, приносит своим жертвам кажущееся освобождение от бремени ответственности. Признание программирования серьёзным вызовом интеллекту вер-

нуло бы полный вес этой ноши обратно на их плечи.

prof. dr. Edsger W. Dijkstra  
Burroughs Research Fellow