1972
Смиренный программист

Эдсгер В. Дейкстра

Отрывок из объявления о награждении Тьюринговской премией, зачитанного М. Д. Макилроем, председателем Комитета по Тьюринговским премиям во время представления настоящей лекции 14 августа 1972 г. на ежегодной конференции ACM в Бостоне:

«Повсюду профессиональный словарь программиста полон слов, введенных или предложенных Э. В. Дейкстрой, — дисплей, мертвая хватка, семафор, программирование с минимумом операторов «go to», структурное программирование. Однако его влияние на программирование является более всепроникающим, чем это мог бы подсказать любой словарь. Ценнейший вклад Дейкстры, признанием которого служит эта Тьюринговская премия, — его стиль: подход к программированию как к высокому искусству и интеллектуальному творчеству, настойчивые требования и практическая демонстрация того, что программы должны быть с самого начала правильно составлены, а не просто отлаживаться до тех пор, пока не станут правильными; ясное понимание того, какие проблемы лежат в основе программирования. Он опубликовал около дюжины статей как технического, так и концептуального характера, среди которых следует особенно отметить его философские обращения к IFIP [1], а также ставшие классическими статьи о взаимодействующих последовательных процессах [2], и его знаменитый обвинительный акт против оператора go to [3]. Недавно появились в виде элегантной монографии оказавшие большое влияние эссе Дейкстры об искусстве составления программ [4]».

Мы научились оценивать хорошие программы так же, как мы оцениваем хорошую литературу. И в центре этого движения находится Э. В. Дейкстра, который создает образцы, столь же прекрасные, как и полезные, и размышляет над ними

Вследствие длинной череды совпадений я официально стал программистом первым весенним утром 1952 г. и, насколько мне удалось выяснить, был первым голландцем, начавшим заниматься этим в моей стране. По прошествии времени я нахожу, что самая поразительная вещь во всем этом, по крайней мере в моей части света, — это та замедленность, с которой появлялась профессия программиста, неторопливость, в которую сейчас трудно поверить. Но, к счастью, в моей памяти сохранилось два ярких воспоминания о том времени, которые не оставляют никаких сомнений в этой неторопливости.

После того как я прозанимался программированием где-то около трех лет, у меня состоялась важная беседа с ван Вейнгаарденом, который был тогда моим руководителем в Амстердамском математическом центре, — беседа, за которую я буду благодарен ему, пока жив. Дело было в том, что, как предпо-

лагалось, я должен был параллельно изучать теоретическую физику в Лейденском университете, но совмещать эти два занятия становилось все труднее, и мне надо было сделать выбор — либо прекратить программировать и стать настоящим респектабельным теоретическим физиком, либо как-то формально завершить мое обучение теоретической физике с минимальными усилиями и стать... кем же? Программистом? Но разве это респектабельная профессия? В конце концов, что такое программирование? В чем должен был состоять тот солидный объем знаний, который позволил бы считать программирование достойной уважения интеллектуальной научной дисциплиной? Я живо вспоминаю, как я завидовал своим коллегам-электронщикам, которые в ответ на вопрос об их профессиональной компетенции могли по меньшей мере указать, что они знают все об электровакуумных приборах, усилителях и прочем, в то время как я чувствовал, что мне нечего ответить на подобный вопрос. Полный мрачных предчувствий, я постучал в дверь кабинета ван Вейнгаардена и спросил, может ли он уделить мне несколько минут для разговора. Когда я покидал его кабинет несколько часов спустя, я был другим человеком. Терпеливо выслушав, что меня заботит, он согласился, что в настоящее время есть не так уже много вещей, которые можно было бы отнести к дисциплине программирования, но затем он спокойно продолжал объяснять, что автоматические вычислительные машины — не кратковременная мода, за ними будущее, что мы находимся у самых истоков и — как знать? — может быть, именно я призван в будущем превратить программирование в почтенную научную дисциплину. Это был поворотный момент всей моей жизни, и я завершил свои занятия теоретической физикой настолько быстро, насколько это было возможно. Из этой истории следует по крайней мере один вывод: давая советы молодежи, нужно быть очень осмотрительным: иногда молодые следуют этим советам!

Два года спустя, в 1957 г., я женился, а в Голландии при регистрации брака необходимо отвечать на вопрос о вашей профессии, и я заявил, что я программист. Но муниципальные власти города Амстердама это не устроило — они считали, что такой профессии не существует. И хотите верьте, хотите нет — в графе «профессия» в моем брачном свидетельстве стоит смехотворная запись: «физик-теоретик»!

Этого достаточно, чтобы показать, насколько медленно профессия программиста завоевывала признание в моей стране. С тех пор я повидал немало других стран, и мое общее впечатление состоит в том, что и там, кроме, возможно, некоторого сдвига по времени, процесс становления этой профессии был очень похожим.

Позвольте мне описать положение в те давние времена несколько подробнее; надеюсь, это поможет лучше понять современную ситуацию. Попутно мы увидим, как много недоразумений относительно истинной природы программирования уходят своими корнями в то теперь уже далекое прошлое.

Все первые автоматические электронные компьютеры были уникальными, построенными в единственном экземпляре машинами, и окружение, в котором все они были сооружены, отличала волнующая атмосфера экспериментальной лаборатории. Как только возникала идея построить автоматический компьютер, ее реализация становилась дерзким вызовом доступной в те времена электронной технологии, и одно можно сказать наверняка: мы не можем отказывать в смелости тем людям, которые решались попробовать создать такую фантастическую технику. А она действительно была фантастической: теперь можно только удивляться, что те первые машины вообще работали, хотя бы изредка. Надо всем царила одна цель — привести компьютер в рабочее состояние и поддерживать его в этом состоянии. Озабоченность физическими аспектами автоматического вычисления до сих пор отражена в названиях самых старых научных обществ в этой области, таких, как Ассоциация вычислительных машин (ACM) или Британское вычислительное общество; в этих названиях прямо содержится упоминание об аппаратной части.

А как же несчастный программист? Честно говоря, его едва замечали. И все потому, что первые машины были настолько громоздки, что их даже невозможно было сдвинуть с места, а кроме того, они требовали такого трудоемкого обслуживания, что было вполне естественно пытаться использовать машину в той же лаборатории, в которой она была разработана. Во-вторых, невидимая в каком-то смысле работа программиста была лишена всякого внешнего блеска: посетителям можно было демонстрировать машину, а это на много порядков превышает по зрелищности какой-то клочок бумаги с программой. Но, что важнее всего, сам программист относился к своей работе как к весьма скромному делу: вся его значительность была связана с существованием этой замечательной машины. Поскольку это была единственная машина, он слишком хорошо знал, что его программа годится лишь для нее одной, и так как было очевидно, что срок существования этой машины ограничен, он знал, что очень немногое в его работе имеет непреходящее значение. И наконец, было еще одно обстоятельство, оказавшее глубокое влияние на отношение программиста к своей работе: с одной стороны, кроме того, что его машина ненадежна, она еще и слишком медленна, а ее память слишком мала, т. е. ресурсов всегда недостаточно, а с другой стороны, ее обычно не-

сколько странный код команд допускал самые неожиданные конструкции. И в те дни многие умные программисты получали огромное интеллектуальное удовольствие от хитрых приемов, посредством которых им удавалось добиваться невозможного, разрешая довольно сложные задачи при всех тех ограничениях, которые налагало оборудование.

С тех времен берут начало два распространенных мнения о природе программирования. Я сейчас их упомяну и вернусь к ним позже. Первое мнение состояло в том, что по-настоящему компетентный программист должен обладать склонностью к разгадыванию головоломок и любить хитроумные уловки; другое в том, что программирование есть не более чем та или иная оптимизация эффективности вычислительного процесса.

Последнее мнение явилось результатом часто возникавшей ситуации: действительно, доступное оборудование нередко сурово ограничивало возможности вычислений, и в те времена часто приходилось сталкиваться с наивной надеждой, что, как только появятся более мощные машины, программирование перестанет быть проблемой, потому что тогда не будет необходимости выжимать из машины все, что она может. А ведь программирование как раз в этом и состоит, не правда ли? Но в следующие десятилетия произошло нечто совсем другое: стали доступными более мощные машины, более мощные даже не на один порядок, а на несколько. Но вместо того, чтобы найти все проблемы программирования решенными раз и навсегда, мы поняли, что наступил настоящий кризис программного обеспечения. Почему же так получилось?

Есть одна несущественная причина: в одном или двух отношениях современная аппаратура принципиально более сложна в обращении, чем старая. Во-первых, появились прерывания ввода-вывода, они происходят непредсказуемо, и моменты их появления невоспроизводимы. По сравнению со старой последовательной машиной, которая по замыслу являлась полностью детерминированным автоматом, такая разница драматична, и седые волосы многих системных программистов свидетельствуют о том, что мы не должны легкомысленно относиться к возникающим при этом логическим проблемам. Во-вторых, современные машины оборудованы многоуровневыми запоминающими устройствами, что заставляет нас думать о стратегии управления, которая, несмотря на обширную литературу на эту тему, остается весьма неясной. Вот, пожалуй, и все, что стоит сказать о структурных новшествах современных машин.

Я назвал все это несущественной причиной, главная же причина в том, что ... машины стали на много порядков более мощными! Подойдем к этому весьма прямолинейно: когда машин не было вовсе, программирование не составляло никакой проб-

лемы; когда у нас было несколько маломощных компьютеров, программирование стало проблемой средней сложности, а теперь, когда мы располагаем гигантскими компьютерами, программирование в свою очередь превращается в гигантскую проблему. В этом смысле электронная промышленность не разрешила ни одной проблемы, она их только создала, а именно проблему использования своей продукции. Другими словами, по мере того как мощность машин возрастает более чем в тысячу раз, честолюбивые замыслы общества, связанные с применением этих машин, растут в той же пропорции, и именно несчастный программист обнаруживает, что его работа оказалась в поле напряжения между целями и средствами. Возросшая мощность аппаратного обеспечения вместе с, возможно, еще более драматичным ростом надежности оборудования, сделали приемлемыми решения, о которых программист раньше не осмеливался и мечтать. А теперь, несколько лет спустя, он вынужден мечтать о них, и, что еще хуже, он должен сделать подобные мечты реальностью! Удивительно ли, что мы пришли к кризису программного обеспечения? Разумеется, нет, и, как легко догадаться, это предсказывалось заранее. Однако с предсказаниями не очень крупных пророков всегда бывает так: лишь через лет пять всем становится ясно, что они были верными.

Тогда, в середине шестидесятых, случилось нечто ужасное: появились компьютеры так называемого третьего поколения. Из официальных документов известно, что отношение цена/качество было одной из главных целей их разработки. Но если считать «качеством» коэффициент использования различных компонентов машины, вы едва ли сумеете избежать такого проекта, в котором «качество» достигается в основном за счет внутренних обменов информацией между компонентами машины, полезность которых сомнительна. А если ваше определение цены означает цену, которую необходимо заплатить за аппаратуру, то у вас, скорее всего, получится такой компьютер, для которого будет очень трудно программировать. Например, набор команд мог бы быть таким, что уже на ранних этапах на программиста или систему будут наложены ограничения, приводящие к неразрешимым на самом деле конфликтам. И похоже, в значительной степени подобные неприятные возможности становятся реальностью.

Когда о таких ЭВМ было объявлено и стали известны их функциональные спецификации, многим из нас, должно быть, стало не по себе; по крайней мере такое чувство возникло у меня. Было естественно ожидать, что такие вычислительные машины хлынут потоком на компьютерный рынок, и, следовательно, тем более важно, чтобы их конструкция была как можно более разумной. Но проект содержал такие серьезные ошиб-

ки, что я почувствовал, что одним ударом прогресс в информатике был заторможен по меньшей мере на десять лет. Это была самая черная неделя во всей моей профессиональной карьере. Быть может, печальнее всего то, что после всех этих лет разочаровывающего опыта многие все еще честно верят в то, что в силу некоторого закона природы машины должны быть именно такими. Они заглушают свои сомнения, видя, как много таких машин было продано, и выводят отсюда, что в конце концов их конструкция не должна быть уж очень скверной. Но при более пристальном рассмотрении подобная линия защиты имеет такую же убедительность, как вывод о том, что курение — здоровое занятие, ведь так много людей курят.

Именно в этой связи я сожалею, что научные журналы в области информатики не имеют обыкновения публиковать обзоры новых проектов компьютеров, вроде того как они печатают обзоры научных публикаций. Обзор вычислительных машин был бы не менее важен. И вот я хотел бы здесь покаяться: в самом начале шестидесятых годов я написал такой обзор и намеревался его отправить в журнал «Communications of ACM», но несмотря на то, что несколько коллег, которым я разослал текст обзора для отзыва, советовали мне это сделать, я не осмелился, боясь, что трудности как для меня, так и для редколлегии журнала могут оказаться слишком большими. Это было с моей стороны актом трусости, и за это я себя ругаю все больше и больше. Те трудности, которые я предвидел, были следствием отсутствия общепризнанных критериев, и хотя я был убежден в справедливости предпочтенных мною критериев, я опасался, что мой обзор не будет принят или будет забракован как «отражающий» лишь личное мнение. Я до сих пор полагаю, что подобные обзоры были бы крайне полезны, и с нетерпением ожидаю, когда они появятся, поскольку их признание было бы верным знаком зрелости программистской общественности.

Причиной, по которой я уделил столько внимания аппаратной части, было ощущение, что одним из наиболее важных аспектов любого вычислительного устройства является его влияние на способ мышления тех, кто его использует. У меня есть причины полагать, что это влияние во много раз сильнее, чем обычно думают. Переключим теперь наше внимание на программное обеспечение.

Вначале была машина EDSAC в Кембридже, Англия, и я считаю очень примечательным, что с самого начала понятие библиотеки подпрограмм играло центральную роль в конструкции этой машины и способе, которым она должна была использоваться. Теперь прошло 25 лет, и все, что связано с компьютерами, изменилось самым драматичным образом, однако понятие базового программного обеспечения до сих пор при нас,

а понятие замкнутой подпрограммы до сих пор одно из ключевых в программировании. Мы должны признать, что замкнутая подпрограмма является одним из величайших изобретений программного обеспечения. Оно пережило три поколения компьютеров и переживет еще больше, потому что воплощает одну из наших фундаментальных абстракций. К сожалению, важность подпрограммы недооценивалась при создании третьего поколения компьютеров, в котором большое число явно присвоенных имен регистров арифметического устройства приводит к большим непроизводительным затратам при реализации подпрограмм. Но даже это не погубило понятие подпрограммы, и нам только остается молиться о том, чтобы эта мутация не передалась по наследству.

Второе крупное преобразование в области программного обеспечения, о котором мне хотелось бы напомнить, — это рождение Фортрана. В те времена это был чрезвычайно смелый проект, и разработавшие его люди заслуживают нашего восхищения. Было бы абсолютно несправедливо винить их в недостатках, которые выявились только примерно после десяти лет его широкого использования: ведь коллективы разработчиков, которым удается предвидеть на десять лет вперед, чрезвычайно редко встречаются! В ретроспективе мы должны оценивать Фортран как успешную методику программирования, но с очень ограниченным числом средств поддержки основной концепции, — средств, которые ныне столь остро необходимы, что пришла пора считать этот язык устаревшим. Чем раньше мы забудем, что Фортран когда-либо существовал, тем лучше, потому что в качестве способа мышления он уже перестал быть адекватным: он заставляет нас напрасно тратить наши мыслительные способности; кроме того, он слишком рискован, и, следовательно, им слишком дорого пользоваться. Трагическая судьба Фортрана — следствие его широкого признания, приковавшего мышление тысяч и тысяч программистов к нашим прошлым ошибкам. Я денно и нощно молю, чтобы как можно больше моих собратьев-программистов нашли способ освободиться от проклятия совместимости.

Третий проект, о котором мне не хотелось бы умолчать, это Лисп, вдохновляющее предприятие совсем другого рода. При небольшом числе весьма фундаментальных принципов он проявил замечательную устойчивость. Кроме того, на использовании Лиспа основаны многие в некотором смысле наиболее изощренные программные продукты. В шутку Лисп описывался как «наиболее интеллигентный способ злоупотребления компьютером». Я думаю, что подобная характеристика является большим комплиментом, поскольку она передает всю полноту освобождения: Лисп помогает многим из наших наиболееода-

ренных коллег мыслить о вещах, ранее считавшихся немыслимыми.

Четвертый проект, который хотелось бы упомянуть, — это Алгол-60. В то время как вплоть до нынешних дней программисты на Фортране склонны понимать свой язык программирования в терминах тех специфических реализаций, над которыми они работают (отсюда и преобладание восьмеричных или шестнадцатеричных распечаток), а определение Лиспа до сих пор остается причудливой мешаниной из того, что язык означает, и того, как он работает, знаменитое Сообщение об алгоритмическом языке Алгол-60 является плодом подлинных усилий перейти на следующий уровень абстрактности и определить язык программирования способом, не зависящим от его реализации. Критически настроенные могли бы сказать, что авторы Сообщения настолько в этом преуспели, что посеяли серьезные сомнения в самой возможности его реализации! Сообщение великолепно продемонстрировало могущество формального метода БНФ, хорошо известной ныне формы Бэкуса — Наура, а также могущество тщательно сформулированной английской прозы, по крайней мере когда ею пользуется кто-то столь же талантливый, как Питер Наур. Я полагал, что справедливости ради следует сказать: только очень небольшое число столь кратких, как этот, документов имели столь глубокое влияние на специалистов по информатике. Пожалуй, сомнительным комплиментом репутации Алгола послужила та легкость, с которой в более поздние времена использовались слова «Алгол» и «алголоподобный»; подобно незащищенному товарному знаку, они одалживали свою славу порой совсем не имеющим к ним отношения незрелым проектам. Сила метода БНФ как инструмента для определения ответственна за то, что я рассматриваю как одну из слабых сторон этого языка: излишне сложный и не слишком систематический синтаксис стало возможным втиснуть в несколько страниц текста определений. Такой мощный инструмент, как БНФ, мог бы сделать сообщение об алгоритмическом языке Алгол-60 намного короче. Кроме того, у меня начали возникать сомнения по поводу механизма параметров в Алголе-60; он дает программисту так много комбинационной свободы, что для того, чтобы программист пользовался им с уверенностью, он должен соблюдать жесткую дисциплину. Кроме того, что его реализация обходится слишком дорого, им, по-видимому, опасно пользоваться.

Наконец, хотя следующий сюжет и не слишком приятен, я должен упомянуть PL/I, язык программирования, для которого определяющая его документация имеет устрашающий объем и сложность. Пользоваться языком PL/I, должно быть, так же сложно, как управлять самолетом с 7000 кнопок, переключате-

лей и рукояток в кабине пилота. Я совершенно неспособен понять, как мы можем надеяться сохранять ориентацию в наших постоянно растущих программах, если из-за своей явной вычурности язык программирования — напомню, что это наш основной инструмент! — уже ускользнул из-под нашего контроля. И если бы мне надо было описать то влияние, которое имеет язык PL/I на своих пользователей, наиболее подходящая метафора, которая приходит мне на ум, — это наркотик. Я вспоминаю одну лекцию на симпозиуме по языкам программирования высокого уровня, прочитанную в защиту языка PL/I человеком, называвшим себя его преданным пользователем. Но в конце этой часовой лекции, восхваляющей PL/I, он ухитрился попросить добавления около 50 новых «свойств», ничуть не догадываясь, что основным источником его трудностей является именно то, что язык уже имеет слишком много «свойств». Лектор проявил все удручающие симптомы наркомании: дошедший до отупения, он мог лишь просить все больше, больше, больше.... В то время как Фортран был назван детской болезнью, весь язык PL/I, растущий с неудержимостью опасной опухоли, может оказаться неизлечимой болезнью.

Вот что было в прошлом. Однако нет никакого смысла делать ошибки, если потом не учиться на них. В действительности же я считаю, что мы столь многому уже научились, что через несколько лет программирование будет настолько существенно отличаться от того, что оно представляло собой до сих пор, что лучше бы было заранее подготовиться к этому удару. Я попробую набросать здесь один из вариантов будущего. При первом взгляде на нее эта картина программирования в недалеком теперь уже будущем может поразить вас как крайне фантастическая. Я хочу добавить несколько соображений, которые помогут поверить в то, что она может стать реальностью.

А картина такова, что еще до завершения семидесятых мы сможем изобрести и реализовать системы такого рода, которые сейчас находятся на пределе наших возможностей программировать, и затратить на них лишь несколько процентов тех усилий, которых они требуют ныне, и, кроме того, эти системы будут практически свободными от ошибок. Эти два усовершенствования будут идти рука об руку. В последнем отношении программное обеспечение, по-видимому, отличается от многих других продуктов, когда более высокое качество, как правило, связано с более высокой ценой. Те, кто желает иметь действительно надежное программное обеспечение, обнаружат, что они для начала должны найти способ избежать большинства ошибок, и как результат, процесс программирования станет дешевле. Если вы желаете иметь более эффективных программистов, вы обнаружите, что они не должны терять время на отладку

и устранение неполадок, они прежде всего не должны делать ошибок. Другими словами, достижение обеих этих целей требует одного и того же изменения подхода.

Такая решительная перемена за такой короткий период времени была бы революцией, и всем, кто основывает свои представления о будущем на гладких экстраполяциях недавнего прошлого, ссылаясь на некие неписаные законы социальной и культурной инерции, вероятность такой решительной перемены может показаться пренебрежимо малой. Но мы все знаем, что революции иногда происходят. Но насколько вероятно, что эта революция произойдет?

По-видимому, надо, чтобы выполнились три основных условия. Весь мир должен признать необходимость перемены; во-вторых, экономическая необходимость этой перемены должна быть достаточно сильной; и, в-третьих, эта перемена должна быть технически выполнимой. Обсудим эти три условия в перечисленном порядке.

Что касается признания необходимости большей надежности программного обеспечения, то я не ожидаю никаких возражений. Но не столь давно дело обстояло иначе: разговор о кризисе программного обеспечения считался кощунством. Поворотной точкой была Конференция по технике программного обеспечения в Гармише в октябре 1968 г. Эта конференция стала сенсационной, когда на ней впервые был открыто признан кризис программного обеспечения. А теперь повсеместно признано, что разработка какой-либо крупной сложной системы является трудным делом и люди, ответственные за такие предприятия, обычно очень озабочены именно вопросами надежности, и это вполне справедливо. Короче говоря, наше первое условие, по-видимому, выполнено.

Что касается экономической необходимости, то прежде можно было зачастую встретиться с мнением, что в шестидесятые годы программистам платили слишком много и в будущем их заработки должны упасть. Обычно это мнение высказывается в связи с экономическим спадом, но оно может быть и симптомом какого-нибудь другого и вполне здорового явления, а именно: программисты прошлого десятилетия, похоже, не столь хорошо выполняли свою работу, как были должны. В обществе нарастает неудовлетворенность качеством работы программистов и их продуктов. Но имеется еще один фактор, существенно более важный. При теперешнем положении дел вполне обычно, что для конкретной системы цена, назначаемая за разработку программного обеспечения, имеет такой же порядок величины, как цена аппаратуры, и общество более или менее спокойно принимает это. Но изготовители аппаратуры сообщают нам, что, как ожидается, в следующем десятилетии цены на

аппаратуру упадут в десять раз. Если разработка программного обеспечения будет оставаться таким же громоздким и дорогостоящим процессом, как ныне, это равновесие цен резко нарушится. Нельзя ожидать, что общество с этим согласится, и, следовательно, нам надо научиться программировать на порядок эффективнее. Иначе говоря, пока вычислительные машины были самой крупной статьей расходов, программистам удавалось получать финансирование, несмотря на их неуклюжую и расточительную методологию, но это прикрытие исчезнет очень быстро. Короче говоря, второе наше условие тоже выполнено.

Теперь рассмотрим третье условие. Является ли такая перемена технически выполнимой? Я думаю, что это возможно, и приведу вам шесть доводов в пользу этого моего мнения.

Изучение структуры программ показало, что программы — даже альтернативные программы для одной и той же задачи и с тем же математическим содержанием — могут очень сильно различаться по своей удобочитаемости и предсказуемости. Было открыто несколько правил, нарушение которых либо серьезно нарушает, либо полностью разрушает удобочитаемость и предсказуемость поведения программы. Имеется два рода правил. Выполнение правил первого рода нетрудно обеспечить механически, т. е. подходящим выбором языка программирования. Примером служит исключение операторов go to и процедур с более чем одним внешним параметром. Что касается правил второго рода, то я по крайней мере — возможно, из-за отсутствия компетентности — не вижу способов их механического установления, поскольку, по-видимому, требуется некое устройство для доказательства теорем; неизвестно, можно ли обеспечить такой механизм. Таким образом, в настоящее время, а может быть, и навсегда, правила второго рода представляют собой элементы той дисциплины, которая требуется от программиста. Некоторые из правил, которые я имею в виду, настолько очевидны, что им легко обучить, и не может возникнуть споров по поводу того, удовлетворяет ли им данная программа или нет. Примерами служат требования, что нельзя написать цикла, не обеспечив доказательства правила остановки или без формулирования соотношения, которое остается неизменным, сколько бы раз не выполнялись повторяемые операторы.

Я предлагаю ограничиться пока разработкой и реализацией удобочитаемых программ с предсказуемым поведением. Если кто-то опасается, что это ограничение слишком сурово и мы не можем с ним мириться, я могу его уверить: класс удобочитаемых и предсказуемых программ все еще достаточно широк и содержит много очень реалистичных программ для любой алгоритмически разрешимой задачи. Мы не должны забывать, что составлять программы — не наше дело: наше дело разра-

батывать классы вычислений, ведущих себя предсказуемым образом. Мое предложение ограничиться удобочитаемыми и предсказуемыми программами — это основа для первых двух из шести объявленных мной доводов.

Первый довод состоит в том, что если программисту приходится рассматривать только программы с предсказуемым поведением, то альтернативы, из которых он выбирает, намного легче оценивать и сравнивать.

Второй довод заключается в том, что, поскольку мы решили ограничиться предсказуемыми программами, мы раз и навсегда добились резкого сокращения рассматриваемого пространства решений. И этот довод отличен от предыдущего.

Третий довод основан на конструктивном подходе к проблеме правильности программы. В настоящее время общепринятой техникой является составление программы, а затем ее тестирование. Однако тестирование программы может быть очень эффективным способом демонстрации наличия ошибок, но оно безнадежно неадекватно для доказательства их отсутствия. Единственный эффективный способ значительно повысить доверие к программе — это дать убедительное доказательство ее правильности. Но не следует сначала писать программу, а потом доказывать ее правильность, поскольку в этом случае требование найти доказательство только увеличит тяготы бедного программиста. Напротив, программист должен доказывать правильность программы одновременно с ее написанием. Третий довод существенно основан на следующем наблюдении. Если прежде всего задать себе вопрос, какова будет структура убедительного доказательства, и только затем строить программу, удовлетворяющую требованиям этого доказательства, то эта озабоченность правильностью оказывается очень эффективным эвристическим руководством. По определению этот подход применим только тогда, когда мы ограничиваемся программами с предсказуемым поведением, но он дает нам эффективное средство нахождения удовлетворительной программы среди множества таких программ.

Четвертый довод относится к способу, которым количество интеллектуальных усилий, необходимых для составления программы, зависит от длины программы. Высказывалось предположение, что существует некий закон природы, гласящий, что количество необходимых интеллектуальных усилий пропорционально квадрату длины программы. Однако, слава богу, никто так и не смог доказать этот закон. Причина в том, что он вовсе не обязательно справедлив. Все мы знаем, что единственное мыслительное средство, посредством которого вполне конечный фрагмент рассуждения может охватывать миллионы случаев, называется «абстракцией». Поэтому наиболее важным видом

деятельности компетентного программиста можно считать эффективную эксплуатацию его способности к абстрагированию. В этой связи может быть полезным подчеркнуть, что назначение абстракции не в том, чтобы быть неясной, но в том, чтобы создавать новый семантический уровень, в котором возможно достичь абсолютной точности. Конечно, я пытался отыскать фундаментальную причину, мешающую нашим механизмам абстрагирования быть достаточно эффективными. Но как бы я не старался, этой причины я найти не смог. Поэтому я склоняюсь к предположению, — до сих пор не опровергнутому опытом, — что при надлежащем применении наших способностей к абстракции интеллектуальное усилие, требующееся для написания или понимания программы, пропорционально не более чем длине самой программы. Побочный продукт этого исследования может иметь гораздо большее практическое значение, и действительно, он является основой для моего четвертого довода. Этим побочным продуктом было установление ряда способов абстрагирования, которые играют важную роль во всем процессе написания программ. Об этих абстрактных конструкциях известно столько, что можно было бы посвятить каждой из них целую лекцию. Какие последствия может иметь знание и сознательное использование этих приемов абстрагирования, открылось мне, когда я понял, что если бы это было известно 15 лет тому назад, то, например, шаг от БНФ к синтаксически ориентированным компиляторам занял бы несколько минут вместо нескольких лет. Следовательно, я выставляю наши новейшие знания о существенных способах абстрагирования в качестве четвертого довода.

Теперь обратимся к пятому доводу. Он имеет отношение к влиянию инструмента, который мы пытаемся использовать, на наши собственные мыслительные привычки. Я заметил некоторую культурную тенденцию, которая, по всей вероятности, своими корнями уходит в эпоху Возрождения, состоящую в том, чтобы не замечать этого влияния, считать человеческий ум высшим и независимым хозяином всего, что он порождает. Но если я начинаю анализировать свои собственные мыслительные привычки, а также и привычки своих коллег, то я независимо от своей воли прихожу совсем к другому выводу, а именно что инструменты, которые мы пытаемся использовать, а также язык или обозначения, применяемые нами для выражения или записи наших мыслей, являются главными факторами, определяющими нашу способность хоть что-то думать или выражать! Анализ влияния, которое имеют языки программирования на мыслительные привычки своих пользователей, и признание того, что теперь именно интеллектуальные ресурсы ограничивают наши возможности более чем что-либо другое, дают нам новый набор

критериев для сравнения относительных достоинств различных языков программирования. Компетентный программист вполне сознает жесткую ограниченность своих способностей; поэтому он подходит к задаче программирования в полном смирении и наряду с некоторыми другими вещами как чумы боится хитроумных трюков. Я слышал со всех сторон о хорошо известном диалоговом языке программирования, что, если программисты оснащены терминалом для него, происходит специфическое явление, известное под названием «однострочники». Оно принимает одну из двух различных форм: один программист кладет перед другим однострочную программу и либо гордо говорит, что она делает, и добавляет вопрос: «Можете закодировать это меньшим числом литер?» — как если бы это имело какое-нибудь значение! — или же просто говорит, «Угадайте, что она делает?» Из этого наблюдения мы должны заключить, что этот язык в качестве инструмента есть открытое приглашение к хитроумным трюкам; и хотя именно это может до некоторой степени объяснить его привлекательность для тех, кто любит демонстрировать свое интеллектуальное превосходство, но, прошу меня простить, я должен рассматривать это как одно из самых тяжких обвинений, которые можно выдвинуть против языка программирования. Другой урок, который мы должны извлечь из недавнего прошлого, — то, что разработка «более богатого» или «более мощного» языка программирования была ошибкой в том смысле, что эти барочные излишества, эти нагромождения вкусовых причуд действительно неудобоваримы и для компьютера, и для человека. Я предвижу большое будущее у очень систематических и очень скромных языков программирования. Когда я говорю «скромный», я имею в виду, что не только, например, от «оператора for» Алгола-60, но и от фортрановского «цикла do», быть может, придется отказаться как от слишком вычурных. Я провел несколько экспериментов с действительно опытными добровольцами-программистами, но произошло нечто совершенно незапланированное и неожиданное. Ни один из моих добровольцев не нашел очевидного и наиболее элегантного решения. При более пристальном изучении оказалось, что причина одна и та же: их понятие об итерации было так тесно связано с идеей дискретного приращения соответствующей управляемой переменной, что в их сознании существовала преграда, не позволяющая им видеть очевидное. Их решения были менее эффективными, неоправданно малопонятными, и их нахождение потребовало слишком много времени. Этот эксперимент меня поразил, но в то же время и многое объяснил. Наконец, в одном отношении есть надежда, что будущие языки программирования будут сильно отличаться от привычных ныне языков: в гораздо большей степени, чем до сих пор, они долж-

ны способствовать отражению в структуре того, что мы записываем, всех абстракций, необходимых для осознания сложности того, что мы разрабатываем. Вот и все о большей адекватности наших будущих инструментов, являющейся основой для пятого довода.

В качестве «реплики в сторону» я хочу добавить предупреждение тем, кто отождествляет сложность задачи программирования с борьбой против неадекватности наших современных инструментов, потому что они могут прийти к выводу, что, как только наши инструменты станут более адекватными, программирование перестанет быть проблемой. Программирование всегда останется очень трудным, поскольку, как только мы освободимся от вызванной несущественными обстоятельствами громоздкости, мы сразу же столкнемся с проблемами, которые пока далеко выходят за рамки того, что мы можем программировать.

Вы можете спорить с моим шестым доводом, поскольку не так легко собрать экспериментальные факты для его подтверждения, однако это не мешает мне верить в его справедливость. До сих пор я еще не упоминал слова «иерархия», но справедливо будет заметить, что это — ключевое понятие для всех систем, реализующих хорошо разбитое на части решение. Я могу даже пойти еще дальше и сделать из этого символ веры: мы в действительности можем решать удовлетворительным образом лишь такие задачи, которые в конечном итоге допускают хорошо разбивающееся на части решение. На первый взгляд эта картина человеческой ограниченности может поразить вас как крайне удручающая и безвыходная, но я так не думаю. Напротив, лучший способ научиться жить с нашей ограниченностью — это знать о ней. В настоящее время мы настолько скромны, что пытаемся находить только решения, разложимые на части, поскольку другие усилия ускользают от нашего понимания. Мы должны делать все возможное, чтобы избежать всех тех взаимодействий, которые подрывают нашу способность разложить систему на составные части полезным образом. И я не могу не ожидать, что это вновь и вновь будет приводить нас к открытию, что первоначально неподдающаяся разложению система в конце концов оказывается разложенной на составные части. Любой, кто видел, как большая часть неприятностей, возникающих на фазе компиляции, называемой «генерация выполнимого кода», может быть прослежена до странных свойств языка, на котором эта программа написана, узнает простой пример того, что я имею в виду. Широкая применимость хорошо разложимых на компоненты решений — это мой шестой и последний довод в пользу технической реализуемости революции, которая может произойти в ближайшие десять лет.

В принципе я предоставляю вам самим решать, сколь много веса вы придаете моим соображениям, поскольку вы прекрасно знаете, что я не могу никого заставить разделять мою веру. Как во всякой серьезной революции, я столкнусь с резкой оппозицией, и можно задать вопрос, откуда следует ожидать появления консервативных сил, пытающихся противодействовать такому ходу событий. Я не ожидаю, что они появятся главным образом из области большого бизнеса, и даже не из компьютерного бизнеса: скорее, я ожидаю их появления со стороны учебных заведений, которые в настоящее время занимаются подготовкой специалистов, а также со стороны пользователей компьютеров, которые находят свои старые программы столь важными, что не считают нужным переписывать или улучшать их. В этой связи, к сожалению, следует заметить, что во многих университетах выбор центральных вычислительных средств очень часто определялся требованиями немногих традиционных, но дорогостоящих приложений вне зависимости от того, сколько тысяч «малых пользователей», желающих написать свои собственные программы, пострадают от этого выбора. Слишком часто, например, физика высоких энергий, как мне кажется, шантажировала научную общественность стоимостью всего своего остального экспериментального оборудования. Самый легкий ответ, конечно, — это отрицание технической реализуемости, но я боюсь, что для этого вам понадобятся слишком сильные аргументы. Увы, нельзя рассчитывать на то, что интеллектуальный «потолок» среднего современного программиста станет препятствием для этой революции: когда другие начнут программировать намного эффективнее, тот, кто не сумел перестроиться, в любом случае окажется оттесненным в сторону.

Могут быть также политические препятствия. Даже если мы знаем, как обучать завтрашнего профессионального программиста, мы вовсе не уверены, что общество, в котором мы живем, позволит нам это сделать. Первым результатом от преподавания методологии — а не распространения знания — является увеличение способностей уже способных, тем самым увеличивается разброс интеллектуальных возможностей. В обществе, в котором система образования используется как инструмент для установления гомогенизированной культуры, в котором сливкам не дают подняться наверх, воспитание компетентных программистов должно быть политически неприемлемым.

Я хочу перейти к заключению. Автоматические компьютеры существуют вот уже четверть века. Они сильно повлияли на наше общество в качестве инструментов, но их воздействие в этом качестве не более чем рябь на поверхности нашей культуры по сравнению с более глубоким влиянием, которое они

окажут в качестве интеллектуального вызова, который не имеет прецедента в культурной истории человечества. По-видимому, иерархические системы обладают тем свойством, что некий объект, рассматриваемый как неделимое единство на некотором уровне, на более низком уровне (с большей степенью детализации) рассматривается как составной объект; в результате естественный масштаб пространства или времени, применимый на каждом уровне, уменьшается на порядок, когда мы переходим с одного уровня на следующий, более низкий. Мы понимаем, что такое стена, представляя ее как способ укладки кирпичей, кирпичи — как способ укладки кристаллов, кристаллы — как способ упаковки молекул и т. д. Поэтому число уровней иерархической системы, которые имеет смысл различать, примерно пропорционально логарифму отношения между самым крупным и самым мелким масштабом, и, следовательно, если только это отношение не слишком велико, мы не должны ожидать слишком большого числа уровней. В программировании на компьютере наш основной строительный «кирпич» имеет соответствующий временной масштаб порядка микросекунды, а наша программа может потребовать для своего выполнения нескольких часов вычислительного времени. Я не знаю другой технологии, охватывающей отношение в 10^{10} раз или более: компьютер благодаря своему фантастическому быстродействию, по-видимому, первый среди устройств, обеспечивающих рабочую среду, в которой многоуровневая иерархия искусственных конструкций не только возможна, но и необходима. Этот вызов, т. е. столкновение с задачей программирования, является столь уникальным, что этот новый опыт может нам открыть глаза на нас самих. Он должен углубить наше понимание процессов проектирования и творчества; он должен научить нас лучше управлять задачей организации нашего мышления. Если он этого не сделает, то, на мой взгляд, мы вовсе недостойны иметь компьютер!

Он уже преподал нам не один урок, и тот, который я хочу особо выделить, состоит в следующем. Мы будем лучше справляться с нашей работой программистов, если только мы будем подходить к этой работе с полным сознанием ее ужасающей сложности, если только мы будем верны скромным и элегантным языкам программирования, если мы будем учитывать природную ограниченность человеческого ума и приниматься за эту работу как Очень Смиренные Программисты.

ЛИТЕРАТУРА

[1] Some meditations on advanced programming, Proceedings of the IFIP Congress 1962, 535—538; Programming considered as a human activity, Proceedings of the IFIP Congress 1965, 213—217.
[2] Solution of a problem in concurrent programming, control, CACM8 (Sept. 1965), 569; The structure of the «THE» multiprogramming system, CACM 11 (May 1968), 341—346.
[3] Go to statement considered harmful, CACM 11 (Mar. 1968), 147—148.
[4] A short introduction to the art of computer programming. Technische Hogeschool, Eindhoven, 1971.

ПОСТСКРИПТУМ

Эдсгер Дейкстра
Факультет информатики
Университет штата Техас, Остин

Моя Тьюринговская лекция 1972 г. в значительной степени представляла собой изложение и обоснование принципиального убеждения: задача, стоящая перед программистом, — это интеллектуальный вызов высочайшего ранга. Меня поражает то обстоятельство, что и по сей день это кредо полностью сохранило свою актуальность: по-прежнему основная трудность информатики состоит в том, как не заблудиться в тех сложностях, которые создаем мы сами.

Однако высказанные в той лекции предложения, как следует отвечать на этот вызов, несомненно принадлежат времени, когда она была прочитана. Если бы я читал ее сегодня, то большую ее часть отвел бы роли формализованных методов программирования.

Сопоставление моих тогдашних ожиданий с тем, что действительно произошло за эти годы, рождает смешанные чувства. С одной стороны, мои самые дерзкие надежды исполнились с лихвой: ясные, четкие аргументы, ведущие к изощренным алгоритмам, которые было очень трудно или невозможно представить себе еще десять лет тому назад, стали постоянным источником интеллектуальных поисков. С другой стороны, меня огорчает, когда я вижу, сколь малая часть этих достижений вошла в обычные учебные курсы информатики, в которых эффективной разработкой высококачественных программ пренебрегают ради всяких модных поветрий (скажем, ради «постепенного самоулучшения дружелюбности интерфейсов экспертных систем»).

Есть какая-то верхняя грань скорости, с которой общество в состоянии усваивать прогресс, и, по-видимому, мне еще нужно учиться, как стать более терпеливым.