Чарльз Энтони Ричард Хоар

Старые платья императора

ACM Turing Award Lectures. The First Twenty Years 1966–1985 // ACM Press, 1987.
Лекции лауреатов премии Тьюринга // М.: Мир, 1993.
А. А. Бряндинская, перевод с англ.

---

27 октября 1980 г. на ежегодной конференции в Нэшвилле (США, шт. Теннесси) председатель комитета по премиям Уолтер Карлсон вручил премию ACM им. Тьюринга (ACM Turing Award) Чарльзу Энтони Ричарду Хоару (Charles Anthony Richard Hoare), профессору вычислительной математики Оксфордского университета в Англии.

Профессор Хоар был избран Комитетом по премиям ACM за свой фундаментальный вклад в определение и создание языков программирования. Его труд отличает неординарное сочетание интуиции, самобытности, элегантности и влияния на развитие компьютерных наук. Наибольшую известность он завоевал благодаря работе над аксиоматическими определениями языков программирования при помощи методов, которые принято называть аксиоматическими семантическими методами. Он разработал искусные алгоритмы, такие как алгоритм быстрой сортировки, а также изобрел и пропагандировал современные методы структурирования данных в языках программирования. Он также внес важный вклад в развитие операционных систем в результате изучения концепции мониторов. Самая последняя его работа посвящена последовательным взаимодействующим процессам (CSP).

До назначения в Оксфордский университет в 1977 г. профессор Хоар был с 1968 по 1977 г. профессором вычислительной математики в Королевском университете в Белфасте (Ирландия) и в 1973 г. читал лекции в Стенфордском университете (США). С 1960 по 1968 гг. он работал на нескольких должностях в фирме Elliott Brothers Ltd. (Великобритания).

Профессор Хоар много публиковался и входит в состав редколлегий нескольких всемирно известных журналов по информатике. В 1973 г. ему присудили премию ACM за лучшую работу по языкам программирования и программным системам. В 1978 г. профессор Хоар стал почетным членом Британского компьютерного общества (British Computer Society, BCS). В 1979 г. ему присудили степень почетного доктора в университете Южной Калифорнии.

Премия Тьюринга — это наивысшая премия Ассоциации вычислительных машин (Association for Computing Machinery, ACM) за большой научный вклад в деятельность вычислительного сообщества. Она вручается каждый год в память д-ра А. М. Тьюринга, английского математика, внесшего большой вклад в развитие компьютерных наук.

Автор делится своим опытом в проведении в жизнь, разработке и стандартизации компьютерных языков программирования и делает одно преостережение на будущее.

Мой первейший и самый приятный долг в этой лекции — выразить глубокую благодарность Ассоциации ACM за большую честь, оказанную мне, и за возможность обратиться к вам с докладом на любую тему по моему выбору. Но какой это трудный выбор! Мои научные достижения, столь полно оцененные этой премией, были уже достаточно описаны в научной литературе. Вместо того чтобы повторять малопонятную специальную терминологию из моей профессиональной области, я хотел бы немного неофициально поговорить о себе, о своем опыте, надеждах и опасениях, скромных успехах и менее скромных неудачах. Из неудач я почерпнул гораздо больше, чем можно рассказать сухим слогом научных статей, и теперь я хочу, чтобы вы тоже кое-чему научились на этих неудачах. Кстати, неудачи выглядят более забавными потом, когда о них рассказываешь, но когда они случаются, они не так уж забавны.

Начало моего рассказа относится к 1960 г., когда я стал программистом в небольшой компьютерной фирме, филиале компании Elliott Brothers Ltd. (Лондон), где в следующие за этим 8 лет я и должен был получить свое первоначальное компьютерное образование. Моим первым заданием было модифицировать для нового компьютера Elliott 803 библиотечную программу нового быстрого метода внутренней сортировки, как раз тогда изобретенного Шеллом. Я с радостью воспринял это как вызов максимизировать эффективность простой программы для

десятично-адресуемой машины тех дней. Моему начальнику и наставнику Пэту Шеклтону очень понравилась составленная мною программа. Тогда я застенчиво сказал, что мне кажется, будто я изобрел метод сортировки, который будет работать быстрее метода Шелла, и при этом он потребует ненамного больше памяти. Он поспорил со мной на шестипенсовик, что я ошибаюсь. Хотя мой метод трудно поддавался объяснению, все же мы пришли к выводу, что я выиграл пари.

Я написал несколько других высококачественных библиотечных подпрограмм, но через полгода мне было дано более важное задание — разработать новый усовершенствованный язык программирования высокого уровня для следующего компьютера компании — Elliott 503, который должен был, имея тот же набор команд, что и предыдущая модель 803, быть в 60 раз более производительным. Несмотря на то что я получил классическое образование и изучал древние языки, это было задание, для которого я обладал даже меньшей квалификацией, чем те, кто берется за эту работу сегодня. По счастливой случайности в мои руки попал экземпляр отчета о международном алгоритмическом языке Алгол-60. Разумеется, этот язык был слишком сложным для наших клиентов. Как могли они разобраться во всех begin и end, если даже наши продавцы не могли?

Примерно на Пасху 1961 г. в Брайтоне (Англия) был введен курс Алгола-60, который преподавали Питер Наур, Эдсгер Дейкстра и Питер Ланден. Я посещал этот курс вместе с моей сотрудницей по языковому проекту Джилл Пим, научным руководителем отдела Роджером Куком и коммерческим директором Полом Кингом. Именно здесь я впервые узнал о рекурсивных процедурах и увидел, как программировать метод сортировки, который я раньше считал трудным для объяснения. Именно здесь я написал программу, нескромно названную «быстрая сортировка» (Quick Sort), которая легла в основу моей карьеры ученого в области компьютеров. Следует отдать должное гению разработчиков Алгола-60 за то, что они включили в свой язык рекурсию и дали мне тем самым возможность весьма элегантно описать мое изобретение. Сделать возможным изящное выражение хороших мыслей — я считал это наивысшей целью проекта языка программирования.

По завершении курса Алгола-60 в Брайтоне, когда Роджер Кук вез нас с коллегами обратно в Лондон, он вдруг спросил: «А что если вместо разработки нового языка мы просто реализуем Алгол-60?» Мы все тут же с этим согласились — я теперь оцениваю это как счастливое для меня решение. Но мы знали в то время, что нам не хватит опыта и сноровки реализовать весь язык, и потому мне предстояло реализовать скромное его подмножество. В этом проекте я принял некоторые основные принципы, которые, я уверен, не утратили своей правильности и по сей день.

(1) Первым принципом была *надежность*. Согласно этому принципу, каждая синтаксически неправильная программа должна быть отброшена компилятором, а на каждую синтаксически корректную программу должен быть выдан результат или сообщение об ошибке, которое должно быть предсказуемым и понятным в терминах текста исходной программы. Следовательно, дампы памяти становились совершенно ненужными. Должно быть логически невозможно для любой программы на входном языке заставить компьютер выйти из-под контроля как во время компиляции, так и во время выполнения. Следствием этого принципа является то, что каждое вхождение каждого индекса каждой индексированной переменной в каждом случае контролируется на выход за объявленные верхнюю и нижнюю границы массива. Много лет спустя мы спросили наших пользователей, хотели ли бы они, чтобы мы предусмотрели опцию транслятора для отключения всех этих проверок с целью повышения скорости выполнения программы. Они единодушно стали нас убеждать, чтобы мы этого не делали — они уже знали, насколько часто происходят ошибки с индексами во время выполнения программ, когда невозможность обнаружить их может оказаться роковой. Я отмечаю с ужасом, что даже в 1980 г. для разработчиков языков и пользователей это не послужило уроком. В любой достойной уважения отрасли техники невыполнение таких элементарных предосторожностей было бы давно уже признано нарушением закона.

(2) Второй принцип в разработке — *краткость объектного кода, порожденного компилятором, и компактность рабочих данных в ходе выполнения*. Причины этого ясны: размеры оперативной памяти любого компьютера ограничены и их расширение предполагает расходы и задержки. Программа, превышающая пределы памяти даже на одно слово, не может работать, в частности, потому, что многие из наших пользователей не намеревались приобретать дополнительные запоминающие устройства. Принцип компактности объектного кода играет не меньшую роль и сейчас, когда процессоры тривиально дешевы по сравнению со стоимостью оперативного запоминающего

устройства, к которому они могут обращаться, а дополнительные запоминающие устройства сравнительно даже еще дороже и медленнее на несколько порядков. Если в результате продуманной реализации доступная аппаратура окажется более мощной, чем может показаться необходимым для конкретного приложения, программист-прикладник почти всегда может воспользоваться дополнительными мощностями для повышения качества его программы, ее простоты, устойчивости и надежности.

(3) Третий принцип при разработке заключается в том, что *входные и выходные соглашения для процедур и функций должны быть столь же компактными и эффективными, как и для написанных в машинном коде подпрограмм*. Я полагал, что процедуры — это одно из самых мощных средств языка высокого уровня, так как они могут одновременно упростить работу программиста и сократить объектный код. Таким образом, не должно быть препятствий для их частого использования.

(4) Четвертый принцип состоял в том, что *компилятор должен быть однопроходным*. Компилятор состоял из набора взаимно рекурсивных процедур, каждая из которых могла анализировать и транслировать основные синтаксические единицы языка — оператор, выражение, объявление и т. д. Он был разработан и документирован в Алголе-60 и затем перекодирован в десятичном машинном коде с использованием явного стека для рекурсии. Без понятия рекурсии в Алголе-60, в то время весьма спорного, мы не смогли бы написать этот компилятор.

Я и теперь могу рекомендовать однопроходный нисходящий рекурсивный спуск и в качестве метода реализации, и в качестве принципа разработки языка программирования. Во-первых, мы, конечно, предполагаем, что программу будут читать люди. А люди предпочитают прочитывать один раз, за один проход. Во-вторых, для пользователей системы с разделением времени или системы персональных компьютеров интервал между вводом программы с помощью клавиатуры (или внесением поправок) и запуском этой программы является полностью непродуктивным. Этот интервал можно минимизировать с помощью быстродействующего однопроходного компилятора. И наконец, придание компилятору структуры, соответствующей синтаксису его входного языка, в большой степени способствует обеспечению его корректности. Если у нас нет полной уверенности в этом, мы никогда не можем доверять результатам какой бы то ни было из наших программ.

Чтобы соблюсти все четыре перечисленных принципа, я выбрал довольно ограниченное подмножество Алгола-60. По мере конструирования и реализации я постепенно обнаруживал методы ослабления этих ограничений, не наносящие урона моим основным принципам. Поэтому в конце работы мы смогли реализовать почти полную мощность этого языка, включая и рекурсию, хотя некоторые его особенности были выброшены, а другие ограничены.

В середине 1963 г. главным образом в результате работы Джилл Пим и Джеффа Холмора была готова первая версия нашего компилятора. Через несколько месяцев у нас стали возникать вопросы, пользуется ли кто-нибудь нашим языком, заметил ли кто-нибудь время от времени выпускаемые нами отчеты, содержащие усовершенствования. Только когда у пользователя возникли претензии, он обращался к нам, но у большинства пользователей претензий не было. Теперь наши пользователи перешли на более современные компьютеры и используют более модные языки, но многие из них признались мне в том, что они с нежностью вспоминают о системе Elliott Algol. И нежность эта связана не просто с ностальгией, но с эффективностью, надежностью и удобством этой прежней алголовской системы.

В результате работы над Алголом в августе 1962 г. я был приглашен в новую рабочую группу 2.1 IFIP, образованную для поддержки и развития Алгола. Первой главной целью этой группы была разработка такого подмножества этого языка, из которого были бы удалены его наименее удачные свойства. Даже в те дни, имея дело с таким простым языком, мы понимали, что подмножество должно быть усовершенствованием исходного языка. Я очень много ждал от возможности встретиться и поговорить со многими мудрыми создателями этого языка. Я был удивлен и шокирован горячностью и даже озлобленностью их споров. Похоже, что первоначальная разработка Алгола-60 не велась в духе бесстрастных поисков истины, как я склонен был предполагать, исходя из его качества.

Чтобы отдохнуть от утомительной и вызывающей разногласия работы по разработке этого подмножества, рабочая группа наметила однажды обсудить те свойства, которые следовало бы включить в следующую версию языка. Каждому члену группы было предложено внести усовершенствование, которое он считал наиболее важным. 11 октября 1963 г. я предложил

перейти к обсуждению просьбы наших пользователей об ослаблении правила, существующего в Алголе-60, обязательно объявлять имена переменных, и о принятии некоторого разумного соглашения по умолчанию, как, например, в Фортране. К моему крайнему удивлению, это, казалось бы невинное, предложение было вежливо, но твердо отклонено: было подчеркнуто, что избыточность Алгола-60 — это его наилучшая защита от ошибок программирования и кодирования, обнаружение которых в уже работающей программе обойдется слишком дорого, а еще дороже — если они не будут обнаружены вовсе. История о том, как космический корабль Mariner, запущенный на Венеру, потерялся из-за отсутствия обязательного объявления в Фортране, была обнародована значительно позже. В конце концов я убедился в необходимости разрабатывать программистские обозначения так, чтобы максимизировать число ошибок, которые невозможно сделать, а если они все же сделаны, то число ошибок, надежно обнаруживаемых в процессе компиляции. Возможно, это увеличило бы длину программы. Но это не имеет значения! Разве не привело бы нас в восторг, если бы добрая фея предложила вам взмахом своей волшебной палочки над вашей программой убрать все ошибки с одним только условием — вы должны переписать и ввести всю вашу программу три раза! Способ сделать программу короче заключается в использовании процедур, а не в опущении жизненно важной информации, содержащейся в явном объявлении имен переменных.

Среди других предложений для развития нового Алгола было такое: switch в Алголе-60 предлагалось заменить более общим средством, а именно массивом переменных, значениями которых являются метки, и так, что программа могла бы менять значения этих переменных с помощью присваивания. Я был категорически против этой идеи, подобной присваиваемой операции GOTO в Фортране, потому что нашел поразительное количество мудреных проблем в реализации даже простых меток и переключателей в Алголе-60. А в этом новом свойстве я усматриваю еще больше проблем, включая такую, как возврат управления назад в блок после того, как он был выполнен. Я к тому же начинал подозревать, что программы, использующие много меток, труднее понимать и отлаживать, а для программ, в которых присваиваются новые значения переменным типа «метка», это сделать еще труднее.

Мне пришло в голову, что подходящая нотация для замены переключателя в Алголе-60 должна быть основана на обозначении условного выражения Алгола-60, которое делает выбор между двумя альтернативными действиями в соответствии со значениями булевского выражения. Другими словами, я предложил обозначения для операторов типа case, которые позволяют выбрать между любым числом альтернатив в зависимости от значения некоторого целого выражения. Это было мое второе предложение в разработке языка. Я до сих пор им горжусь, потому что в сущности оно совсем не создает проблем ни для разработчика, ни для программиста, ни для человека, читающего программу. Ныне, когда прошло более 15 лет, это обозначение было включено в международный нормативный документ по языку — исключительно *короткий* период внедрения по сравнению с другими областями техники.

Но вернемся к моей работе в фирме Elliott. После неожиданного успеха нашего компилятора Алгола все наши помыслы занял более честолюбивый проект: создать операционную систему для более крупной конфигурации компьютера Elliott 503, с устройством для считывания с перфокарт, построчными печатающими устройствами, магнитными лентами и даже с дополнительным запоминающим устройством на магнитных сердечниках, вдвое более дешевым и вдвое более емким, чем оперативное запоминающее устройство, но в 15 раз более медленным. Это то, что позже стало называться системой программного обеспечения Elliott 503 Mark II.

Она включала в себя следующие компоненты:

(1) Транслятор с языка ассемблера, на котором было на писано все остальное программное обеспечение.

(2) Схему автоматического управления оверлейными режимами для кодов и данных, как для накопителей на магнитных лентах, так и для дополнительного запоминающего устройства на магнитных сердечниках. Эта схема должна была использоваться остальным программным обеспечением.

(3) Схему автоматической буферизации всех потоков ввода и вывода на любом доступном периферийном устройстве — снова для использования всем программным обеспечением.

(4) Файловую систему на магнитной ленте с возможностью для редактирования и управления заданиями.

(5) Полностью новую реализацию Алгола-60, лишенную всех нестандартных ограничений, которые мы ввели в нашей первой реализации.

(6) Компилятор с версии Фортрана, распространенной в то время.

Я писал документы, описывающие относящиеся к делу понятия и средства, и мы рассылали их существующим и потенциальным пользователям. В начале работы в нашей команде было 15 программистов, был назначен срок поставки: примерно через полтора года, в марте 1965 г. После начала разработки программного обеспечения Mark II я был внезапно повышен в должности и назначен помощником главного инженера, отвечающего за развитие и разработку аппаратных и программных продуктов фирмы.

Хотя в качестве администратора я был все еще ответствен за программное обеспечение Mark II, я уделял ему меньше внимания, чем новым продуктам фирмы, и почти что прозевал тот момент, когда истек срок поставки этого продукта, а ничего еще не было готово. Программисты пересмотрели свои графики реализации, и в июне 1965 г. была установлена новая дата поставки — через три месяца. Не стоит и говорить, что и эта дата прошла, а результатов все не было. К этому моменту наши клиенты начали сердиться, и мое начальство предложило взять мне этот проект под свою ответственность. Я попросил старших программистов снова подготовить исправленные трафики, и снова оказалось, что проект сможет быть закончен через три месяца. Я очень хотел в это верить, но все же не смог. Я перестал обращать внимание на графики и углубился в проект.

Оказалось, что мы не составили никаких общих планов распределения нашего самого ограниченного ресурса — оперативной памяти. Каждый программист рассчитывал, что это будет сделано автоматически — либо ассемблером, либо автоматической схемой поддержки оверлеев. Хуже того, мы даже не рассчитали, какое пространство занимает наше собственное программное обеспечение, которым уже была заполнена вся оперативная память компьютера, что не оставляло места для работы программ наших пользователей. Ограничения на длину аппаратных адресов не позволяли увеличить оперативную память.

Стало ясно, что исходные технические требования к програминому обеспечению не могли быть удовлетворены, и их следовало коренным образом урезать. Опытные программисты и даже менеджеры были освобождены от других проектов. Мы решили прежде всего сосредоточиться на поставке нового компилятора для Алгола-60, что должно было по самым тщательным подсчетам занять еще четыре месяца. Я постарался внушить всем программистам, занятым в проекте, что это больше не предсказание, а обещание; если они посчитают, что не могут выполнить этого обещания, они сами отвечают за то, чтобы найти пути и способы для выполнения работы.

Реакция на этот вызов со стороны программистов была замечательной. Они работали дни и ночи, чтобы обеспечить завершение всех тех элементов программного обеспечения, которые требовались для компилирующей программы на Алголе. К нашему восторгу, они уложились в назначенные сроки; это был первый важный фрагмент рабочего программного обеспечения, произведенный компанией за двухлетний период. Но восторг наш длился недолго: оказалось, что компилятор не может быть поставлен. Его скорость компилирования не превышала двух литер в секунду, что выглядело очень невыигрышно в сравнении с уже существующими компиляторами, работавшими со скоростью тысячи литер в секунду. Мы тотчас же нашли причину: она заключалась в метании между оперативной памятью и расширением — дополнительным запоминающим устройством на магнитных сердечниках, которое работало в 15 раз медленнее. Нетрудно было внести несколько простых усовершенствований, и через неделю скорость компилятора удвоилась до четырех литер в секунду. В следующие две недели, посвященные исследованиям и перепрограммированиям, скорость снова была удвоена до восьми литер в секунду. Нам уже было ясно, как за месяц добиться дальнейшего улучшения; но число требуемых перепрограммирований увеличивалось, а их эффективность уменьшалась; это был слишком длинный путь. Альтернатива, состоящая в увеличении размера оперативной памяти, так часто выбираемая впоследствии при неудачах подобного рода, была невозможна из-за ограничений на адресацию аппаратного характера.

Выхода не было: надо было отказываться от всего проекта разработки программного обеспечения Elliott 503 Mark II, а вместе с ним выбросить в корзину около 30 человеко-лет программистской работы, что эквивалентно почти целому сроку активной деятельности человека за всю его жизнь, и я нес ответственность и как разработчик, и как менеджер за их напрасную трату.

Было созвано заседание всех наших клиентов, работающих на модели Elliott 503, и Роджер Кук, бывший тогда менеджером отдела вычислений, объяснил им, что они никогда не получат ни одной команды столь долго ожидаемого программного обеспечения. Он докладывал об этом спокойным ровным голосом, это не позволяло никому из пользователей его прерывать, шептаться на задних рядах или даже ерзать на своих местах. Я смотрел на него с восхищением,

но не мог разделять его спокойствия. Во время ланча наши пользователи были столь любезны, что пытались утешить меня. Они давно уже поняли, что программное обеспечение, соответствующее исходным техническим условиям, никогда не может быть поставлено, а если бы и было, то клиенты не знали бы, как воспользоваться их столь изощренными свойствами, и, во всяком случае, много столь же крупных проектов было аннулировано до поставки. Теперь, глядя в прошлое, я верю, что нашим клиентам повезло, что ограничения, вносимые аппаратурой, уберегли их от неограниченного потока наших разработок в области программного обеспечения. В наши дни пользователи микропроцессоров пользуются подобной же защитой — но это ненадолго.

В то время я прочитал документы, описывающие концепцию и свойства новой операционной системы IBM OS/360 и новый проект ОС разделения времени, названный Multics. Эти документы по своей понятности, продуманности и сложности превышали все, что я мог себе представить даже в первой версии программного обеспечения модели Elliott 503 Mark II. Очевидно, IBM и MIT обладали каким-то секретом успешной разработки и реализации программного обеспечения, о природе которого я даже не пытался догадываться. Лишь позже эти фирмы осознали, что им он также неизвестен.

Таким образом, я до сих пор не могут понять, как мне удалось навлечь такое несчастье на мою фирму. В то время я был убежден, что мои начальники собираются меня уволить. Но они намеревались меня наказать еще более строго. «Хорошо, Тони,— сказали они.— Вы втянули нас в такие неприятности, а теперь вы нас вытащите из них». «Но я не знаю, как» — протестовал я. «Ну тогда вам придется в этом разобраться». Они даже выразили уверенность, что я справлюсь. Я этой уверенности не разделял. Я был склонен отказаться. Из всех моих удач самая большая — то, что я этого не сделал.

Разумеется, компания делала все, что было в ее силах, чтобы мне помочь. Они освободили меня от забот об аппаратных разработках и сократили численность моих программистских групп. Каждый из моих менеджеров изложил свою собственную теорию, объясняющую причину неудач, и у всех были разные теории. В конце концов они пригласили в мой офис самого старшего менеджера, генерального директора корпорации, в которую входила наша компания, Эндрю Сент-Джонстона. Я был удивлен тем, что он что-то слышал обо мне. «Знаете, что у вас там произошло? — закричал он, он всегда кричал.— Вы позволили вашим программистам делать вещи, которых вы сами не понимаете». Я смотрел на него с удивлением. Очевидно, он был совершенно незнаком с современным положением вещей. Как один человек может целиком понимать, любой современный программный продукт, такой, например, как система Elliott 503 Mark II?

Позже я понял, что он был абсолютно прав: он правильно диагностировал сущность проблемы и посеял семена правильного решения.

У меня по-прежнему была группа приблизительно из 40 программистов, и нам необходимо было удержать доверие клиентов нашей новой модели и даже снова завоевать его для нашей старой модели. Но что мы должны были планировать, если мы знали только одно — что все наши предыдущие планы потерпели неудачу? Тем не менее 25 октября 1965 г. я устроил продолжавшееся весь день совещание всех своих старших программистов, чтобы выяснить все накопившиеся недоразумения между нами. Я до сих пор храню свои заметки об этом совещании. Прежде всего мы перечислили основные претензии наших клиентов: отмена продуктов, неспособность уложиться в сроки, чрезмерный объем программного обеспечения «...не оправданный полезностью предусматриваемых средств», крайне медленные программы, неумение учитывать обратную связь с пользователем. «Своевременный учет самых скромных требований наших клиентов принес бы не меньше дивидендов в смысле благожелательности пользователей, чем успех наших самых честолюбивых планов».

Затем мы перечислили наши претензии: недостаток машинного времени для тестирования программ, отсутствие четкого графика выделения машинного времени, нехватка подходящего периферийного оборудования, ненадежность аппаратуры, даже если она и доступна, рассредоточенность программистских кадров, нехватка оборудования для клавишного перфорирования программ, отсутствие четких сроков поставки аппаратуры, нехватка технического персонала для документирования, недостаточность знаний в области программного обеспечения за пределами группы программистов, вмешательство вышестоящих администраторов, навязывающих свои решения «...без полного понимания менее очевидных последствий этих решений», а также излишний оптимизм под давлением клиентов и отдела сбыта.

Но мы не стремились оправдать этими претензиями нашу неудачу. Например, мы согласились с тем, что долг программистов — повышение научного уровня своих администраторов и других отделов компании путем «...представления необходимой информации в простой осязаемой форме». Надежда на то, что «...недостатки спецификаций оригинальной программы можно было бы компенсировать мастерством отдела технической документации... не оправдались; разработка программы и разработка ее спецификации должны производиться параллельно одним и тем же лицом, и должно существовать взаимодействие между этими этапами работы. Недостаточная ясность спецификации является вернейшим признаком недостатков в программе, которую она описывает, и эти две ошибки должны устраняться одновременно, до того как приступают к проекту». Жаль, что я не следовал этому совету в 1963 г.; хотелось бы, чтобы все мы следовали ему теперь.

Мои заметки об однодневном совещании в октябре 1965 г. включали в себя целый раздел, посвященный неудачам группы программного обеспечения; этот раздел может соперничать с наиболее униженной самокритикой ревизионистов времен китайской культурной революции. Нашей главной ошибкой было излишнее честолюбие. «Очевидно, что цели, которых мы пытались достичь, оказались намного выше наших возможностей». Мы потерпели также неудачу в прогнозе размеров и быстродействия программ в оценке того, какие требуются усилия: неудачу в планировании координации и взаимодействия программ между собой; нашей ошибкой было то, что мы своевременно не обнаружили, что дела идут плохо. Обнаружились также ошибки в контроле за изменениями в программах, в документировании, связи с другими отделами, с нашим начальством и с клиентами. Мы не смогли дать четкие и неизменяемые определения того, что должен сделать каждый программист и руководитель проекта; стоит ли продолжать? Что больше всего удивляет, так это то, что большая команда весьма умных и знающих программистов смогла так тяжело и так долго работать над таким малообещающим проектом. Известно, что не следует доверять нам, умным программистам. Мы можем выдумать достаточно хорошие доводы, чтобы убедить самих себя и других в какой угодно чепухе. Особенно не верьте нам, когда мы обещаем повторить прежний успех в следующий раз, только увеличив и улучшив его.

Последний раздел нашего расследования неудач содержал критерии качества программного обеспечения. «В нынешней борьбе за то, чтобы выдать хоть какое-нибудь программное обеспечение, первой жертвой оказывается качество готового программного продукта. Качество программного обеспечения оценивается несколькими несовместимыми в полной мере критериями, которые надо тщательно уравновесить при разработке и реализации каждой программы». Затем мы составили список, в который вошло не менее семнадцати критериев, который опубликовали в редакционной статье журнала «Software — Practice and Experience».

Как удалось нам встать на ноги после этой катастрофы? Во-первых, мы разбили клиентов модели Elliott 503 на несколько групп в соответствии с характером и объемом аппаратных конфигураций, которые они приобрели, — например, все приобретшие оборудование с магнитными лентами были отнесены к одной группе. К каждой группе пользователей мы прикрепили небольшую команду программистов и предложили руководителям команд посетить клиентов и разобраться, чего они хотят, выбрать требования, которые проще всего удовлетворить, а затем наметить планы (но не давая обещаний) их выполнения. Ни в коем случае нельзя рассматривать требования, удовлетворение которых заняло бы более трех месяцев. Руководитель проекта должен был убедить меня, что требование клиентов разумно, что разработка нового свойства нужна и что планы и график работ по реализации реалистичны. Самое главное, я не позволял делать ничего, чего бы сам не понимал. И это подействовало! Запросы на программное обеспечение стали выполняться вовремя. Благодаря увеличению доверия между нами и нашими клиентами мы смогли приступить к выполнению более сложных требований. Через год мы оправились от потрясения. Через два года у нас даже были умеренно удовлетворенные клиенты.

Таким образом, благодаря здравому смыслу и компромиссам мы добились чего-то похожего на успех. Но я не был удовлетворен. Я не мог понять, почему разработка и реализация операционной системы должны быть настолько труднее разработки и реализации компилятора. Именно поэтому я посвятил мои последующие исследования проблемам параллельного программирования и языковым конструкциям, которые должны были способствовать четкому структурированию операционных систем — таким, как мониторы и взаимодействующие процессы.

Когда я работал в компании Elliott, я сильно заинтересовался методами формального определения языков программирования. В то время Питер Ландин и Кристофер Стречи предложили определить язык программирования с помощью простой функциональной нотации, которая бы определила результат выполнения каждой команды на математически определенной абстрактной машине. Мне не очень нравилось это предложение, поскольку я чувствовал, что такое определение должно включать в себя ряд вполне произвольных решений способа

представления, и в принципе оно должно быть ненамного проще, чем реализация языка на реальной машине. В качестве альтернативы я предложил, чтобы определение языка программирования было формализовано в виде последовательности, аксиом, описывающих желаемые свойства программ, написанных на этом языке. Мне казалось, что тщательно сформулированные аксиомы оставляют достаточно свободы для эффективной реализации языка на разных машинах и позволяют программисту доказать правильность его программ. Но как это сделать, мне было неясно. Я думал, что потребуются длительные исследования для разработки и применения необходимых методов, а самым лучшим местом для проведения таких исследований является университет, а не промышленность. Таким образом, я обратился на кафедру информатики Королевского университета Белфаста, где мне предстояло провести девять счастливых и продуктивных лет. В октябре 1968 г., когда я распаковывал свои бумаги в новом доме в Белфасте, я наткнулся на препринт малоизвестной статьи Боба Флойда, озаглавленной «Assigning Meanings to Programs» (Придание смысла программам). Вот это была удача! Наконец-то я увидел, как достигнуть того, к чему я стремился в моих исследованиях. Тогда я написал свою первую статью об аксиоматическом подходе к программированию на компьютере, которая была опубликована в Communications of the ASM в октябре 1969 г.

Как раз незадолго до этого я обнаружил, что одним из первых, кто отстаивал метод проверки программы с помощью утверждений, был никто иной, как сам Алан Тьюринг. В июне 1950 г. на конференции в Кембридже он прочел короткое сообщение, озаглавленное «Проверка большой программы», в которой эта идея очень ясно объяснялась. «Как можно проверить большую программу, чтобы убедиться, что она верна? Чтобы облегчить задачу проверяющего, программист должен сделать некоторое число определенных *утверждений*, которые можно было бы проверить по отдельности и из правильности которых следовала бы правильность всей программы».

Рассмотрим аналогичный случай — проверку сложения. Если дана сумма (столбец цифр с ответом внизу), то надо проверять все за один раз. Если же даны суммы для нескольких столбцов (просуммированных по отдельности), работа проверяющего облегчается, поскольку она разбивается на проверки нескольких различных утверждений (т. е. что каждый столбец правильно просуммирован) и небольшое суммирование (чтобы получить итоговую сумму). Этот принцип можно применить к проверке большой программы, но мы проиллюстрируем этот метод маленькой программой, а именно получение n-факториала без помощи устройства умножения. К несчастью, не существует достаточно широко известной системы кодирования, чтобы можно было оправдать приведение здесь всей программы, но для иллюстрации достаточно блок-схемы. Это возвращает меня к основной теме моего доклада, к разработке языков программирования.

С августа 1962 по октябрь 1966 г. я бывал на каждом собрании рабочей группы по Алголу при IFIP. После завершения нашей работы по подмножеству Алгола IFIP мы начали разрабатывать Algol-X, который намеревались сделать преемником Алгола-60. Были выдвинуты дополнительные предложения о включении в язык новых свойств, и в мае 1965 г. Никлаусу Вирту было поручено собрать их все и разработать единый проект языка. Я пришел в восторг от его чернового проекта, которому удалось избежать всех известных недостатков Алгола-60 и который содержал несколько новых свойств. Все они могли быть легко и эффективно реализованы, а их применение было надежным и удобным.

Описание этого языка все еще не было закончено. Я много поработал над его улучшением, и многие члены нашей бригады тоже. Ко времени следующего совещания в октябре 1965 г. во Франции, в местечке Сент-Пьер де Шартрез, у нас был черновой вариант замечательного и реалистичного проекта языка, который был опубликован в июне 1966 г. под названием «Вклад в развитие Алгола» в Communications of the ACM. Он был реализован на IBM 360 и получил название Algol-W, которое ему дали его многочисленные счастливые пользователи. Это был не только удачный преемник Алгола-60, это был к тому же удачный предшественник Паскаля.

На том же самом собрании комитету по Алголу был предложен краткий, неполный и довольно невразумительный документ, описывающий другой, более претенциозный и, по моему мнению, куда менее привлекательный язык. Я был поражен, когда рабочая группа, состоящая из всех наиболее известных международных экспертов по языкам программирования, решила оставить в стороне черновой проект, над которым мы все работали, и заглотнула такую неаппетитную приманку.

Это произошло как раз через неделю после нашего заключения о проекте программного обеспечения Elliott 503 Mark II. Я выразил отчаянное предостережение против нечеткости, сложности и чрезмерной претенциозности нового проекта, однако мой голос не был услышан. Я пришел к заключению, что существуют два способа составления проекта программного

обеспечения: один способ — сделать его таким простым, чтобы было *очевидно*, что недостатков нет, а другой — сделать его таким сложным, чтобы не было *очевидных* недостатков.

Первый метод намного труднее. Он требует такой же смекалки, преданности делу, проницаемости и даже вдохновения, как открытие простых физических законов, лежащих в основе сложных явлений природы. Это также требует готовности принять цели, ограниченные физическими, логическими и технологическими требованиями, и согласиться на компромисс, когда нельзя достигнуть согласия между противоречивыми требованиями. Ни один комитет не сможет добиться этого своевременно, согласие достигается лишь тогда, когда все сроки уже прошли.

Именно это и случилось с комитетом по Алголу. Было ясно, что проект, который они предпочли, не был совершенным. Поэтому было обещано, что новый и окончательный проект нового языка Алгол будет готов через три месяца. Он должен, был быть затем тщательно изучен подгруппой из четырех членов группы, в которую входил и я. Три месяца прошли, но ни слова о новом проекте не было слышно. Через полгода подгруппа устроила совещание в Нидерландах. У нас был более длинный и более толстый документ, полный исправленных в последнюю минуту ошибок, описывающий еще один новый, но для меня столь же непривлекательный язык. Никлаус Вирт и я провели некоторое время, пытаясь устранить некоторые недостатки в проекте и в описании, но тщетно. Завершенный окончательный набросок языка был обещан на следующей встрече всего комитета по Алголу, намеченной через три месяца.

Снова прошли три месяца — и ни слова о новом наброске не появилось. Через полгода, в октябре 1966 г., рабочая группа по Алголу собралась в Варшаве. Она рассматривала еще более толстый и длинный документ, пестрящий исправленными в последнюю минуту ошибками, описывающий еще один, столь же темный, а для меня столь же непривлекательный язык. Эксперты группы не смогли увидеть недостатков проекта и твердо решили принять набросок в надежде, что он будет закончен через три месяца. Напрасно я говорил им, что он не будет готов. Напрасно я настаивал на устранении технических ошибок языка, преобладания ссылок, преобразований типа по умолчанию. Рабочая группа отнюдь не стремилась к упрощению языка и требовала от авторов, чтобы они включили еще более сложные свойства, такие, как совмещение операторов и параллелизм.

Когда какой-нибудь проект нового языка близится к завершению, всегда возникает безумная спешка — внести новые свойства еще до стандартизации. Это стремление действительно безумно, потому что оно приводит в ловушку, из которой нет выхода. Недостающие свойства всегда можно добавить позже, когда их конструкция и их последствия будут хорошо поняты. Свойство, включенное прежде, чем оно было понято, никогда нельзя изъять позже.

Наконец в декабре 1968 г., настроенный крайне пессимистически, я поехал на встречу в Мюнхен, где наш долго вынашиваемый монстр должен был появиться на свет и получить имя Алгол-68. К тому времени некоторые другие члены группы также разочаровались в проекте, но было слишком поздно: теперь комитет был заполнен сторонниками этого языка и проект был направлен для утверждения вышестоящим органам IFIP. Все, что мы смогли сделать, — отправить вместе с ним заявление меньшинства, формулирующее наше твердое убеждение, что «... как инструмент для надежного создания сложных программ данный язык непригоден». Этот отчет был позднее скрыт — акт, который напомнил мне строчки из Хилэра Беллока:

> Ученые мужи — не верить им грешно!
> Нас уверяют, что так быть должно.
> Не дай нам Бог сомненье допустить
> В том, в чем никто себя не в силах убедить.

Я больше не был ни на одном собрании этой рабочей группы. Я с удовольствием сообщаю, что вскоре эта группа пришла к пониманию, что с этим языком, а также с его описанием не все в порядке; они напряженно проработали еще шесть лет, чтобы создать пересмотренное описание этого языка. Это было значительным усовершенствованием, однако боюсь, что, на мой взгляд, оно не устранило основных технических изъянов проекта, а также полностью проигнорировало проблему чрезмерной сложности этого языка.

Программистам всегда приходится соприкасаться со сложностью; этого нельзя избежать. Наши приложения сложны, потому что мы честолюбивы и стремимся использовать наши компьютеры все более сложными способами. Программирование сложно из-за большого числа противоречивых целей, преследуемых каждым программным проектом. Если наш основной инструмент, язык, на котором мы составляем и кодируем программу, тоже непрост, то сам язык становится частью нашей проблемы, а не ее решения.

А теперь я расскажу вам о другом излишне амбициозном языковом проекте. Между 1965 и 1970 гг. я состоял членом и даже председателем технического комитета № 10 Европейской ассоциации производителей компьютеров (ECMA). Сначала нам было предложено рассмотреть краткое резюме, а затем стандартизировать некий язык, который должен был вытеснить все языки и был предназначен для всех приложений компьютеров, как научных, так и коммерческих; инициатива исходила от крупнейшего изготовителя компьютеров всех времен. С интересом и удивлением и даже с некоторым удовольствием я изучил четыре первоначальных документа, описывающие язык NPL, появившиеся в период между мартом и ноябрем 1964 г. Из них каждый следующий был более претенциозным и абсурдным, чем предыдущий, в своих обещаниях и желании превзойти все известное ранее. Затем язык начали реализовывать, с интервалами в полгода появлялись все новые документы, и каждый из них описывал новый окончательный вариант этого языка под его окончательным названием PL/I.

Но, на мой взгляд, каждый пересмотр документа просто показывал, как далеко продвинулась реализация первоначального замысла. Те компоненты языка, которые еще не были реализованы, по-прежнему описывались свободной цветистой прозой, обещающей ничем не омраченное удовольствие. А в тех компонентах, которые уже были реализованы, цветы уже поблекли: их заглушила поросль объяснительных примечаний, накладывающих произвольные и неприятные ограничения на использование каждого свойства и налагающих на программиста ответственность за отслеживание сложных и неожиданных побочных эффектов и эффектов взаимодействия с другими свойствами языка.

Наконец, 11 марта 1968 г. описание языка было честь честью представлено с нетерпением его ожидающей научной общественности в качестве достойного кандидата на стандартизацию. Но он им не был. Это описание уже подверглось семи тысячам исправлений и модификаций его авторов и создателей. Понадобилось еще двенадцать изданий, прежде чем он был окончательно опубликован в качестве стандарта в 1976 г. Я боюсь, что это произошло не потому, что все причастные к этому проекту люди были удовлетворены своей работой, а потому, что им все это надоело и они расстались с иллюзиями.

Все то время, пока я хоть как-то участвовал в этом проекте, я не уставал добиваться, чтобы язык был упрощен; если это необходимо, хотя бы выделением ограниченного его подмножества, которое профессиональный программист смог бы понять и отвечать за правильность, стоимость и эффективность своих программ. Я настаивал на том, чтобы опасные свойства, такие, как соглашения по умолчанию и ON-условия, были устранены. Я знал, что будет невозможно написать полностью надежный компилятор для столь сложного языка, когда правильность каждой части программы зависит от проверки того, чтобы все другие части этой программы избежали всех ловушек и ошибок языка.

Вначале я надеялся, что такой технически неразумный проект потерпит крах, но вскоре я понял, что он обречен на успех. Почти все в программном обеспечении может быть реализовано, продано и даже использовано, если проявить достаточную настойчивость. Ничто из того, что может утверждать какой-то там ученый, не может остановить поток сотен миллионов долларов. Но существует одно качество, которое нельзя купить таким образом,— это надежность. Цена надежности — это погоня за крайней простотой. Это цена, которую очень богатому труднее всего заплатить.

Все это произошло много лет тому назад. Можно ли считать, что все это имеет отношение к конференции, посвященной предвидению компьютерной эры, на пороге которой мы стоим? Больше всего я боюсь, что имеет, и притом самое непосредственное. Ошибки, которые мы совершили за последние 20 лет, продолжают повторять ныне и даже в больших масштабах. Я имею в виду проект разработки языка, который породил документы, озаглавленные «соломенный человек», «деревянный человек», «жестяной человек», «железный человек», «стальной человек», «зеленый» и, наконец, Ada. Этот проект был задуман и финансирован одной из самых могущественных организаций мира, Министерством обороны США. Таким образом, ему обеспечено влияние и внимание независимо от его технических достоинств, а его ошибки и дефекты угрожают нам гораздо большими опасностями. Ни одно из имеющихся пока свидетельств не может внушить нам уверенность в том, что в этом языке удалось избежать тех проблем, с которыми сталкивались другие сложные языковые проекты прошлого.

Я высказывал все, какие мог, советы по этому проекту начиная с 1975 г. Вначале я был очень оптимистичен. Первоначальные цели разработки включали надежность, читаемость программ, формализованность определения языка и даже простоту. Постепенно эти цели приносились в жертву мощности, якобы достигнутой благодаря обилию свойств и соглашений по обозначениям, многие из которых были необязательными, а некоторые из них, такие, как обработка

исключительных ситуаций, даже опасными. Обратимся к истории конструирования автомобиля. Мелкие и не очень нужные приспособления преобладают над соображениями безопасности и экономичности.

Но еще не поздно! Я верю, что с помощью старательного удаления лишнего из языка Ada все еще возможно выбрать очень мощное подмножество, которое было бы надежно и эффективно при реализации, а также безопасно и экономично в использовании. Спонсоры этого языка высказывались недвусмысленно, что никаких подмножеств не будет. Это самый странный парадокс этого странного проекта. Если вы хотите, чтобы у языка не было подмножеств, создавайте маленький язык.

Вы включаете только те свойства, о которых вы знаете, что они необходимы для *каждого* приложения языка, и о которых вы знаете, что они годятся для *каждой* аппаратной конфигурации, на которой этот язык реализован. Тогда там, где необходимо, должны быть разработаны расширения для конкретных аппаратных устройств и для конкретных приложений. Великая сила Паскаля в том и состоит, что в нем очень мало ненужных свойств и почти нет нужды в подмножествах. Вот почему этот язык достаточно силен, чтобы выдержать специализированные расширения — Concurrent Pascal для работы в реальном времени, Pascal+ для моделирования дискретных событий и UCSD-Pascal для микропроцессорных рабочих станций. Если бы только мы могли извлекать правильные уроки из прошлых успехов, нам не было бы нужды учиться на наших неудачах.

Итак, лучший из моих советов организаторам и разработчикам языка Ada остался без внимания. И вот, как последнее средство, я обращаюсь к вам, представителям программистской общественности США, а также гражданам, причастным к благосостоянию и безопасности вашей страны и всего человечества: не позволяйте этот язык в его теперешнем состоянии использовать в приложениях, надежность которых критична, таких, как атомные электростанции, крылатые ракеты, системы раннего оповещения, системы противоракетной обороны. Следующая ракета, которая собьется с пути в результате ошибки в языке программирования, может оказаться не исследовательским космическим зондом, летящим в безобидное путешествие к Венере; это может быть ракета с ядерной боеголовкой, которая способна взорваться над одним из наших собственных городов. Ненадежный язык программирования, порождающий ненадежные программы, представляют для окружающей среды и нашего общества несравненно больший риск, чем небезопасные автомобили, токсические пестициды или аварии на атомной электростанции. Неусыпно добивайтесь снижения риска, а не увеличения его.

Позвольте мне закончить не на такой мрачной ноте. Видеть, как твои лучшие советы игнорируются, — такова судьба всех, кто берет на себя роль консультанта, еще с тех пор, когда Кассандра предупредила, что опасно ввозить деревянного коня внутрь стен Трои. Это напомнило мне одну историю, которую я слышал в детстве. Насколько я помню, она называлась так:

Старые платья императора

Много лет тому назад жил один император, который так любил одеваться, что тратил все свои деньги на платья. Он не интересовался армией, не устраивал пиров, не вершил судьбы в суде. О других королях или императорах можно было сказать: «Он заседает в совете», но об этом всегда говорили: «Император сидит в своем гардеробе». Так оно и было. В один несчастный день он был обманут и вышел к народу голым — к своему огорчению и к восторгу своих поданных. Он решил никогда не покидать свой трон и, чтобы не показаться снова перед народом голым, приказал, чтобы каждое из его многочисленных новых одеяний просто надевалось сверху на прежнее.

Время проходило весело в его огромном столичном городе. Министры и придворные, ткачи и портные, гости и поданные, швеи и вышивальщицы заходили и выходили из тронного зала, каждый по своим делам, и все они восклицали: «Как прекрасно одеяние нашего императора!»

Однажды старейший и самый верный министр императора услышал об искусном портном, который закончил старейшую высшую школу швейного мастерства и разработал новое искусство абстрактной вышивки, в которой использовались столь, тонкие стежки, что никто не мог с уверенностью сказать, были ли они или нет. «Должно быть, это действительно замечательные стежки, — подумал министр. — Если бы мы только смогли пригласить этого портного в советники, мы достигли бы в украшении нашего императора таких высот восхваления, что весь мир признал бы его величайшим из всех императоров».

Итак, старый честный министр нанял портного за большое вознаграждение. Портного привели в тронный зал, и он сделал почтительный поклон груде одежды, которая теперь полностью

покрывала весь трон. Все придворные с нетерпением ждали его советов. Представьте себе их изумление, когда он посоветовал не добавить изысканности и более сложной вышивки к ранее существующей, а посоветовал убрать слои пышных нарядов, стремиться к простоте и элегантности вместо экстравагантной утонченности. «Этот портной не такой уж знаток, как он утверждает, — бормотали они. — Его мозги протухли от долгого созерцания его башни из слоновой кости, и он больше не понимает портновских потребностей современного императора». Портной долго и громко отстаивал здравомыслие своего совета, но не мог добиться, чтобы его послушались. В конце концов он получил гонорар и вернулся в свою башню из слоновой кости.

Никогда, вплоть до этого самого дня, не была поведана вся правда о той истории. А она такова. Однажды утром, когда императору стало жарко и скучно, он старательно высвободился из-под груды одежд и зажил счастливо, как свинопас из другой истории. Портной был канонизирован как святой патрон всех консультантов, потому что, несмотря на огромный гонорар, который он получил, он так и не смог убедить своих клиентов в том, что он давно начал подозревать: императора в одеждах нет.

Об авторе. Тони Хоар начал свое карьеру в 1960 г. в Англии, где он писал программы для компании Elliott Brothers Ltd. За восемь лет, проведенных им в этой фирме, он занимался разработкой всевозможных программных продуктов, от простых подпрограмм до языков программирования высокого уровня. Хоар покинул деловой мир в 1968 г., чтобы преподавать информатику в Королевской Университете в Белфасте, Ирландия. В 1977 г. он перешел на факультет информатики Оксфордского университета, где занимает должность профессора вычислительной математики. В 1980 г. был награжден Премией Тьюринга за свой вклад в формальное определение языков программирования посредством аксиоматической семантики. Превращение программирования в серьезную профессиональную дисциплину стало ведущим мотивом его научной деятельности.