Никлаус Вирт

От разработки языка программирования к созданию компьютера

ACM Turing Award Lectures. The First Twenty Years 1966–1985 // ACM Press, 1987.
Р. Богатырев, перевод с англ.

Никлаус Вирт (Niklaus Wirth) из Швейцарского федерального технологического института (ETH) был награжден премией Алана Тьюринга (Turing Award) в 1984 г. на ежегодной конференции Ассоциации ACM (Association for Computing Machinery), состоявшейся в октябре в Сан-Франциско. Это награждение стало признанием его выдающихся заслуг в области разработки ряда новейших языков программирования: Euler, Algol-W, Modula-2 и Паскаль. В частности, язык Паскаль приобрел особое значение в сфере образования и заложил базу для будущих исследований в области языков программирования, систем и архитектуры. Отличительными чертами разработанных Виртом языков являются простота, экономичность и высокое качество проработки деталей, что в конечном итоге приводит к языкам, система обозначений которых скорее является продолжением алгоритмического образа мышления, а не чуждого ему формализма.

Талант Вирта как разработчика языков программирования дополняется писательским даром. В апрельском номере 1971 г. журнала Communications of the ACM Никлаус Вирт поместил основополагающую работу по структурному программированию (Program Development by Stepwise Refinement, "Разработка программы методом поэтапного усовершенствования"), которая рекомендовала нисходящее, идущее сверху вниз, построение программы (т.е. последовательное уточнение фрагментов программы, пока она не окажется полностью переработанной). Полученный в результате элегантный и мощный метод описания представляет интерес и сегодня, даже несмотря на то, что восторги относительно структурного программирования поубавились. Две более поздние статьи "О дисциплине программирования в реальном времени" (Toward a Discipline of Real-Time Programming) и "Что мы можем сделать с необязательным разнообразием обозначений" (What Can We Do about the Unnecessary Diversity of Notation for Syntactic Definitions?), опубликованные в том же журнале соответственно в августе и ноябре 1977 г., говорят о последовательном и неотступном поиске Виртом адекватного языкового формализма.

Премия Тьюринга, являющаяся высшим признанием со стороны Ассоциации ACM вклада в дело компьютерного сообщества, учреждена в честь английского математика Алана М. Тьюринга, создавшего прототип компьютера — машину Тьюринга — и помогшего раскрыть шифры Германии во время Второй мировой войны.

Вирт получил степень доктора философии в Калифорнийском университете в Беркли в 1963 г. и был приглашенным профессором в Стенфордском университете до 1967 г. С 1968 г. он является профессором Швейцарского федерального технологического института в Цюрихе; с 1982 г. по 1984 г. он возглавлял в этом же институте факультет компьютерных наук. Последняя работа Вирта связана с конструированием и разработкой персонального компьютера Lilith в сочетании с использованием языка Modula-2. В своей лекции Вирт кратко излагает историю своих основных проектов, описывает их результаты и формулирует принципы, которые направляли его работу.

Путь, пройденный Виртом в поиске приемлемого формализма системного программирования, начиная с NELIAC, через Алгол-60 к языкам Euler и Algol-W, Паскаль, Modula-2 и, в конечном итоге, до Lilith, полон впечатляющих открытий и удивительных результатов.

Получить премию Тьюринга мне доставляет огромное удовольствие, а признание работы, которая проводилась в течение стольких лет, одновременно и радует, и вдохновляет. Я хочу поблагодарить ACM за присуждение мне этой престижной награды. Особенно важно, что я получаю ее в Сан-Франциско, где начиналась моя профессиональная карьера.

Вскоре после того, как я узнал о присуждении премии, мое чувство радости было несколько омрачено необходимостью представить Тьюринговскую лекцию. У человека, являющегося прежде всего инженером, а не оратором и не проповедником, эта обязанность не может не вызвать заметного беспокойства. Главный среди возникающих вопросов — чего в первую очередь ждут люди от такой лекции? Одни хотят проникнуть в суть чьей-либо работы, услышать оценку ее важности или ожидаемого эффекта. Другие же стремятся узнать, как возникли идеи, лежащие в

ее основе. Третьи ждут авторитетного мнения относительно будущих тенденций, событий и разного рода продуктов. Четвертые надеятся услышать откровенную оценку современного положения вещей: восторги, вызванные огромным прогрессом науки, или же сетования на негативные побочные эффекты и преувеличения.

В нерешительности я просмотрел некоторые из предыдущих Тьюринговских лекций и пришел к выводу, что наиболее приемлемым выходом будет лаконичное сообщение об истории какой-либо работы. Чтобы это не было пустым развлечением, я попытался подвести итог тому, чего мне удалось добиться за последние годы. Откровенно говоря, этот выбор вполне устраивает меня, поскольку я ни в коей мере не претендую на то, что знаю о будущем больше большинства здесь присутствующих, к тому же совершенно не хочу оказаться впоследствии неправым. В то же время искусство читать проведи о нынешних достижениях и прегрешениях отнюдь не является моим коньком. Это не означает, что я равнодушно наблюдаю за происходящим в области компьютерных наук, в частности, за шумной борьбой с коммерциализацией.

Когда в 1960 г. я вступил в область компьютерных наук, им не уделялось столь большого внимания ни в коммерческой рекламе, ни в академических учебных планах. Во время моего обучения в Швейцарском федеральном технологическом институте (ETH) единственное упоминание о компьютерах, которое я услышал, прозвучало на факультативном курсе, читавшемся Амброзом Спайзером (Ambros P. Speiser), ставшим позднее президентом Международной Федерации по обработке информации (IFIP). Разработанный им компьютер ERMETH был малодоступен обычным студентам, и поэтому мое посвящение в компьютерные науки оказалось отложенным до того момента, как я прослушал курс численного анализа в Лавальском университете в Канаде. Но, увы, компьютер Alvac III E большую часть времени был неисправен, и упражнения по программированию так и остались на бумаге в виде непроверенных последовательностей шестинадцатеричных кодов.

Моя следующая попытка оказалась несколько более удачной. В Беркли я столкнулся с любимцем Гарри Хаски (Harry Huskey) — компьютером "Bendix G-15". Хотя этот компьютер позволил почувствовать вкус успеха, поскольку на нем удалось-таки получить хоть какие-то результаты, все искусство программирования представало, как удачное размещение команд на запоминающем барабане. Если вы пренебрегали этим искусством, то ваши программы вполне могли работать в 100 раз медленнее. Однако его учебная ценность была налицо: вы не могли позволить себе игнорировать самую незначительную из мелких деталей. Такого простого способа компенсации недоработок программы за счет приобретения дополнительной памяти тогда просто не существовало. Теперь приятно вспомнить, что каждая деталь машины была видна, и ее назначение можно было понять. Ничего не было скрыто в сложной электронной схеме, в кристаллах кремния или в загадочной операционной системе.

С другой стороны было очевидно, что программирование будущих компьютеров должно быть куда более эффективным. Поэтому я отказался от идеи изучить вначале, как проектировать аппаратную часть, а попытался научиться более элегантно ее использовать. Мне повезло, что я присоединился к группе, участвовавшей в разработке — или, скорее, в доработке — компилятора и в его использовании на IBM 704. Этот язык был назван NELIAC, то был диалект языка Algol-58. Преимущества такого "языка" быстро стали очевидны, а задача автоматической трансляции программ в машинный код поставила трудные вопросы. Это была как раз та ситуация, к которой стремятся при подготовке диссертации. Компилятор, который тоже был написан на языке NELIAC, представлял собой нечто крайне запутанное. Казалось, что он состоит на 1% из науки, а на 99% — из колдовства. Этот перекос и предстояло ликвидировать. Ясно, что программы должны строиться в соответствии с теми же принципами, что и электронные схемы, четко разбивающиеся на блоки, границы которых пересекают всего несколько проводов. Только разобравшись в том, как действует каждая часть в отдельности, можно надеяться в конце концов понять, как действует все это вместе.

Благодаря появлению официального сообщения по языку Алгол-60 эта попытка получила хороший импульс. Алгол-60 стал четко специфицированным языком, а его синтаксис даже определялся в рамках строгого формализма. Урок, который удалось из этого извлечь, состоял в том, что ясное определение является необходимым, но недостаточным условием надежной и эффективной реализации. Контакт с Адрианом ван Вейгаарденом (Aadrian van Wijngaarden) — одним из разработчиков Алгола — позволил более четко прояснить центральный вопрос: можно ли принципы Алгола сконденсировать и выкристаллизовать в еще большей степени?

Вот так и начались мои приключения в мире языков программирования, напоминавшие больше путешествие с мачете сквозь джунгли особенностей и средств языка. Первый эксперимент

привел к диссертации и к языку Euler. Результат оказался академически элегантным, но имел весьма малую практическую ценность — он был почти антитезой более поздним языкам с типами данных и языкам структурного программирования. Но он в действительности заложил фундамент систематической разработки компиляторов, позволявших без потери ясности включать новые возможности.

Язык Euler привлек внимание Рабочей группы Международной Федерации по обработке информации (IFIP), участвовавшей в составлении планов относительно будущего Алгола. Язык Алгол-60, созданный специалистами в области численных методов для своих конкретных целей, обладал систематической структурой и четким определением, что было по достоинству оценено математически подготовленными людьми. Однако ему не доставало компиляторов и поддержки со стороны промышленности. Чтобы завоевать эту поддержку, необходимо было расширить сферу его применения. Рабочая группа взяла на себя задачу предложить преемника этого языка и вскоре распалась на два лагеря. Один состоял из честолюбивых людей, желавших заложить еще один краеугольный камень в фундамент разработки языков, а по другую сторону находились те, кто чувствовал, что время торопит и что должным образом расширенный Алгол-60 может оказаться вполне продуктивным. Я относился ко второму лагерю и выдвинул предложение, которое не нашло у группы достаточной поддержки. Впоследствии это предложение было улучшено с помощью Тони Хоара (Tony Hoare), члена той же группы, и реализовано на первой IBM 360 Стенфордского университета. Позже этот язык стал известен под именем Algol-W и использовался в учебных целях в нескольких университетах.

Стоит упомянуть и небольшое затишье в этой масштабной разработке. Новая IBM 360 предлагала только ассемблер и, естественно, Фортран. Ни то, ни другое ни у меня, ни у моих студентов не вызывало особой симпатии в качестве инструмента для создания компилятора. Поэтому я набрался смелости ввести еще один новый язык, на котором мог бы быть описан компилятор Алгола: представляя собой некий компромисс между Алголом и средствами, предлагаемыми ассемблером, он должен был служить машинным языком, структура операторов и описание команд которого напоминают Алгол. Примечательно, что описание языка было подготовлено всего за пару недель; я написал кросс-компилятор для компьютера Burroughs B-5000 за четыре месяца, а один прилежный студент примерно за такой же отрезок времени переписал его для IBM 360. Это подготовительная пауза помогла существенно ускорить работу с Алголом. Хотя первоначально считалось, что новый язык будет использоваться только для наших насущных потребностей и потом будет прочно забыт, он все же быстро приобрел собственное значение. Язык PL360 стал эффективным инструментом во многих областях и вдохновил на осуществление аналогичных разработок для других машин.

По иронии судьбы успех PL360 вместе с тем говорил о неудаче языка Algol-W. Диапазон применений Алгола был расширен, но в качестве инструмента для системного программирования он все еще грешил явными недостатками. Как мы убедились, трудно удовлетворить сразу стольким требованиям с помощью одного только языка, да и поставленная цель оказалась под вопросом. Язык PL/I, созданный приблизительно в это же время, представил дополнительные доказательства в пользу этой точки зрения. Нет слов, идея, использованная в швейцарском армейском ноже, обладает рядом достоинств, но если переборщить, то этот нож превратится в обузу. Ко всему прочему размер компилятора для языка Algol-W вышел за рамки комфортного ощущения того, что ты еще в силах понять всю программу в целом. Стремление к еще более четкому и в то же время адекватному формализму системного программирования оказалось нереализованным. Для системного программирования необходим эффективный компилятор, генерирующий эффективный код, который работает без фиксированного, скрытого и большого по объему пакета программ так называемого времени выполнения (run-time). Достичь этого не удалось ни на Алголе, ни на PL/I, поскольку оба языка были сложными, а компьютеры, для которых они писались, им не подходили.

Осенью 1967 г. я вернулся в Швейцарию. Еще через год я смог собрать группу из трех помощников для внедрения нового языка, ставшего позднее известным как Паскаль (Pascal). Избавленный от необходимости получать единогласную поддержку Комитета, я смог сосредоточить все свое внимание на том, чтобы включить те характеристики, которые счел существенными, и выбросить те, которые, на мой взгляд, не окупали усилий по их реализации. Да, в некоторых ситуациях жестко ограниченное число сотрудников является безусловным преимуществом.

Высказывалось мнение, что Паскаль был разработан в качестве языка для обучения. Хотя это утверждение справедливо, но использование языка при обучении не было единственной целью. На самом деле я не верю в успешность применения во время обучения таких инструментов и

формальных средств, которые нельзя использовать при решении каких-то практических задач. По сегодняшним меркам Паскаль обладал явными недостатками при программировании больших систем, но 15 лет назад он представлял собой разумный компромисс между тем, чего хотелось, и тем, что было на самом деле эффективным. В 1972 г. в Швейцарском федеральном технологическом институте мы начали использовать Паскаль на занятиях по программированию, правда, встретив при этом серьезное сопротивление. Наш шаг оказался успешным, поскольку он позволил преподавателям уделять больше внимания конструкциям и концепциям, а не отдельным особенностям и характеристикам, другими словами, концентрироваться на принципах, а не технической стороне.

Наш первый компилятор Паскаля был реализован на семействе компьютеров CDC 6000 и сам был написан на Паскале. Никакого PL6000 не потребовалось, и я рассматривал это как существенный шаг вперед. Тем не менее, генерируемый код был, без сомнения, хуже кода, формируемого компиляторами Фортрана для соответствующих программ. Скорость является критерием важным и легко поддающимся количественной оценке, и мы считаем, что обоснованность концепции языка высокого уровня может быть воспринята промышленностью только в том случае, если потери эффективности скомпилированных кодов будут устранены полностью или по крайней мере сведены к минимуму. С учетом этого следующим шагом, причем предпринятым одним человеком, стала попытка создать высококачественный компилятор. Эта цель была достигнута в 1974 г. Урсом Амманом (Urs Ammann). Его компилятор впоследствии получил широкое распространение и продолжает использоваться сегодня многими университетами и организациями промышленности. И все же цена оставалась высокой; усилия по генерации хорошего (даже не оптимального) кода пропорциональны несоответствию языка машине, а CDC 6000, конечно, не была спроектирована с расчетом на языки высокого уровня.

И опять главный выигрыш проявился там, где мы его менее всего ожидали. После того, как стало известно о существовании Паскаля, несколько человек попросили нас помочь в реализации Паскаля на различных машинах, подчеркивая, что они намерены использовать его для обучения, и быстродействие не имеет первостепенного значения. После этого мы решили создать версию компилятора, которая генерировала бы код для машины нашей собственной конструкции. Позднее этот код стал известен как P-код. Создать эту версию компилятора было очень легко, поскольку новый компилятор создавался в качестве важного эксперимента в области структурного программирования по принципу последовательного уточнения, когда первые несколько шагов уточнений могут приниматься без изменений. Pascal-P оказался исключительно удачным для распространения языка среди большого числа пользователей. Если бы у нас хватило мудрости предвидеть масштабы такого развития событий, то мы приложили бы больше усилий и тщательности при разработке и документировании P-кода. А тогда это представляло собой просто побочную деятельность, которой мы занимались только для того, чтобы удовлетворить запросы одним волевым усилием. Это хорошая иллюстрация того, что даже с наилучшими намерениями можно выбрать неверные цели.

Подлинно широкое признание Паскаль получил только после того, как Кен Боулес (Ken Bowles) в Сан-Диего обнаружил, что P-система с успехом может быть реализована на новых микрокомпьютерах. Его усилия по разработке подходящей среды с интегрированным компилятором, навигатором файлов, редактором и отладчиком привели к прорыву: Паскаль стал доступен тысячам пользователей новых компьютеров. Эти пользователи не были обременены ранее приобретенными привычками, и их не душила необходимость сохранять совместимость со старым программным обеспечением.

В это же время я закончил работу над Паскалем и решил заняться изучением нового заманчивого предмета — мультипрограммирования, для которого Хоар уже заложил солидный фундамент, а Бринк Хансен (Brinch Hansen) с помощью своего *Параллельного Паскаля* (Concurrent Pascal) наметил путь реализации. Попытка задать конкретные правила дисциплины мультипрограммирования быстро заставила меня сформулировать их в терминах небольшого набора средств программирования. Чтобы подвергнуть эти правила настоящей проверке, я встроил их во фрагментарный язык, название которого отражало мою главную цель — модульный принцип организации программных систем. Позднее эта модульность оказалась основным достоинством данного языка: она позволила придать абстрактной концепции сокрытия информации конкретную форму и воплотила метод, одинаково важный как для обычного программирования, так и для мультипрограммирования. Помимо того, язык *Modula* также содержал средства для описания параллельных процессов и их синхронизации.

К 1976 г. мне уже надоели и языки программирования, и угнетающая обязанность писать хорошие компиляторы для существующих компьютеров, разработанных для устаревшего

"ручного" кодирования. К счастью, мне представилась возможность провести годичный отпуск, предназначенный для научной работы, в исследовательской лаборатории корпорации Xerox в Пало-Альто, где не только родилась, но и нашла свое практическое воплощение идея мощных персональных рабочих станций. Вместо того, чтобы делить с многочисленными пользователями один большой компьютер и сражаться за свою долю ресурсов, используя кабель с 3-килогерцевой полосой, я теперь через 15-мегагерцевый канал пользовался своим собственным компьютером, находившимся под моим рабочим столом. Последствия увеличения скорости в 5000 раз предвидеть невозможно, слишком оно велико. Самым приятным было то, что после 16 лет моей работы на компьютеры теперь, похоже, компьютер работал на меня. Впервые я вместо того, чтобы намечать планы создания новых языков, компиляторов и программ, которыми будут пользоваться другие, обрабатывал свою ежедневную почту и готовил доклад с помощью компьютера. Другим открытием стало то, что на такой рабочей стадии можно было реализовать компилятор для языка *Mesa*, сложность которого значительно превышала сложность компилятора для Паскаля. Эти новые условия работы на столько порядков превосходили все то, с чем я сталкивался дома, что я решил попытаться создать такие условия и там.

В конечном итоге я решил углубиться в разработку аппаратной части. Это решение подкреплялось моей давней неприязнью к существующим архитектурам компьютеров, которые отравляют жизнь разработчикам компиляторов, склонным к систематической простоте. Идея целиком разработать и построить компьютерную систему, состоящую из аппаратной части, микрокода, компилятора, операционной системы и программных средств, быстро обрела в моем воображении конкретные очертания конструкции, которая была бы свободна от любых ограничений вроде совместимости с PDP-11 или IBM 360, или Фортраном, Паскалем, UNIX'ом и какими бы там еще ни было сиюминутными увлечениями и стандартами Комитета.

Однако для успешной реализации технического проекта одного чувства свободы недостаточно. Упорная работа, уверенность, тонкое ощущение того, что является важным, а что — эфемерным, а также некоторое везение просто необходимы. Первой удачей стал телефонный звонок одного разработчика аппаратной части, интересовавшегося возможностью приехать в наш университет для обучения в области разработки программного обеспечения и получения докторской степени. Почему бу не поучить его разработке программного обеспечения, чтобы он, в свою очередь, поучил нас, как разрабатывать аппаратное обеспечение? Вскоре мы превратились в одну команду, а Ричард Оран (Richard Ohran) так заинтересовался новым проектом, что практически полностью забыл как о программном обеспечении, так и о докторской степени. Это не слишком встревожило меня, поскольку я был достаточно занят конструированием частей аппаратного обеспечения, спецификациями микро- и макрокода, программированием интерпретатора макропрограмм, планированием полной системы программного обеспечения. Я был занят, в частности, программированием редактора текстов и графического редактора, которые оба использовали новый растровый дисплей с высоким разрешением и маленькое чудо под названием "мышь" в качестве координатно-указательного устройства. Этот опыт написания высокоинтерактивных сервисных программ потребовал изучения и использования методов, совершенно чуждых обычному проектированию компиляторов и операционных систем.

В целом проект был столь разносторонним и сложным, что приниматься за него казалось безответственным, особенно учитывая, что число наших помощников, лишь часть своего рабочего времени отводивших этой разработке, было невелико — около семи человек. Основная угроза заключалась в том, что продолжительность этой работы могла оказаться слишком большой, чтобы энтузиазм сохранился у нас двоих и позволил остальным, еще не испытавшим на себе огромных возможностей рабочей станции, стать такими же энтузиастами. Чтобы проект не превысил разумных размеров, я остановился на трех жестких требованиях: задача — создать *однопроцессорный компьютер*, с которым работает *один пользователь* и программирование которого ведется на *одном языке*. Заметим, что эти главные положения были диаметрально противоположны существовавшим тенденциям, направленным на исследования по мультипроцессорным конфигурациям, операционным системам с разделением времени для большого числа пользователей и для такого числа языков, которые только удастся найти.

Ограничившись одним языком, я столкнулся с трудным выбором, который мог иметь самые серьезные последствия, а именно с выбором языка. Из существующих языков ни один не выглядел привлекательным. Они не могли удовлетворить всем требованиям, и к тому же ни в коей мере не устраивали разработчика компилятора, который знает, что задание должно быть выполнено в разумные сроки. В частности, язык должен был удовлетворять всем нашим положениям в отношении структурного программирования, основанным на десятилетнем опыте работы с Паскалем, и быть пригодным для тех задач, которые прежде решались исключительно при кодировании на ассемблере. Короче говоря, было решено создать потомка сразу двух

языков: как уже апробированного Паскаля, так и экспериментального языка Modula; им стал язык *Modula-2*. Модульность — решающее свойство, позволяющее сочетать противоречивые требования: обеспечение надежности абстракции высокого уровня через проверку на избыточность и наличие средств низкого уровня, обеспечивающих доступ к индивидуальным конструктивным особенностям конкретного компьютера. Это дает возможность программисту локализовать использование средств низкого уровня в нескольких небольших частях системы, защищаясь таким образом от непредвиденных осложнений.

Проект *Lilith* доказал, что разработка одноязыковой системы не только возможна, но и обладает рядом преимуществ. Буквально все, начиная с драйверов для устройств и кончая графическим редактором и редактором текста, пишется на одном и том же языке. Никак не отличается подход к модулям, относящимся к операционной системе, и к модулям программы пользователя. Фактически это различие почти пропадает, и вместе с ним мы избавляемся от неизменного громоздкого резидентного блока программы, без которого всякий рад обойтись, но все вынуждены его использовать. Кроме того, проект Lilith доказал преимущества хорошей сочетаемости программного и аппаратного обеспечения. Эти преимущества можно оценить в терминах быстродействия: сравнение времен выполнения программ, написанных на Modula-2, показало, что система Lilith часто предпочтительней, чем система VAX-750, сложность и стоимость которой многократно превышает сложность и стоимость проекта Lilith. Эти преимущества можно также оценить в терминах требуемого объема памяти: машинные коды программ, написанных на Modula-2 для Lilith, в 2-3 раза короче, чем для PDP-11, VAX или Motorola 68000, и в 1,5-2 раза короче, чем для National Semiconductor 32000. Кроме того, части компилятора, генерирующие код, для этих микропроцессоров значительно запутаннее, чем аналогичные элементы системы Lilith, из-за неудачного набора команд. Этот фактор длины кода надо умножить на низкий коэффициент плотности, который омрачает сильно разрекламированную пригодность языка высокого уровня для современных микропроцессоров и показывает, что эти авансы несколько преувеличены. Перспектива того, что такие устройства будут выпущены миллионными сериями, весьма удручает, поскольку благодаря одному их количеству они станут стандартной элементной базой. К сожалению, технология полупроводников развивалась столь быстро, что достижения в области компьютерной архитектуры оказались в тени и кажутся не столь важными. Конкуренция заставляет производителей "замораживать" новые разработки в виде серийно выпускаемых микросхем задолго до того, как они доказали свою эффективность. И в то время как громоздкое программное обеспечение может быть в худшем случае модифицировано, а лучшем случае заменено, сегодня вся сложность спустилась непосредственно на уровень микросхем. Маловероятно, что мы лучше справляемся с вопросами сложности на уровне аппаратного обеспечения, чем на уровня программного.

И среди программистов, и среди инженеров-электронщиков найдется немало людей, для которых сложность есть и будет крайне притягательна. Действительно, мы живем в сложном мире и стараемся решать сложные по своей сути проблемы, которые часто для своего решения требуют сложных устройств. Однако это не значит, что мы не должны стремиться найти *элегантные* решения, убеждающие своей ясностью и эффективностью. Простые элегантные решения более эффективны, но найти их *труднее*, чем сложные, и для этого требуется больше времени. Слишком часто мы считаем, что не можем себе позволить таких затрат.

Прежде чем закончить, я попытаюсь выделить некоторые общие характеристики упомянутых мною проектов. Очень важный метод, который редко используется столь же эффективно, как при вычислениях, это *раскрутка* (bootstrapping). Мы использовали его почти в каждом проекте. При разработке любой системы, будь то язык программирования, компилятор или компьютер, я проектирую ее таким образом, чтобы она была полезной уже непосредственно на следующем этапе: PL360 был разработан для реализации Algol-W, Паскаль — для реализации Паскаля, Modula-2 — для реализации программного обеспечения целой рабочей станции, Lilith — для обеспечения подходящей среды для всех наших будущих работ, начиная от написания программ и до документирования и разработки схемы, от подготовки отчета до проектирования шрифтов. С помощью раскрутки можно извлечь максимальную выгоду из затраченных усилий, но и сильнее всего пострадать от совершенных ошибок.

Все это вынуждает на *раннем этапе различать, что важно, а что второстепенно*. Я всегда пытался выделить существенные моменты, которые дадут несомненные преимущества, и сосредоточиться на них. Например, включение программирования четкой и согласованной схемы объявления типа данных я считаю существенным, в то время как все мелочи, относящиеся к разнообразию циклов, или вопрос о том, должен ли компилятор различать прописные и строчные буквы, являлись для меня второстепенными. При проектировании компьютера я считал

решающим выбор режимов адресации и обеспечение полными и согласованными наборами арифметических операций (со знаками или без знака), включающих прерывания по переполнению, а детали механизма приоритета прерываний при многоканальной системе — второстепенными. Еще важнее гарантия того, что решение второстепенных вопросов никогда не нарушит систематическое структурированное проектирование центральных устройств. Вместо этого второстепенные моменты должны подгоняться под существующую хорошо структурированную основу.

Иногда трудно устоять против настойчивых требований пользователей включить все типы средств, которые "хорошо было бы иметь". Опасность состоит в том, что стремление угодить чьему-то желанию нарушит согласованность всего проекта. Я всегда пытался сбалансировать выгоду и затраты. Например, размышляя, не включить ли некую языковую конструкцию или специальную обработку в компиляторе какой-то достаточно часто используемой конструкции, следует оценить выгоды и дополнительные расходы при реализации, поскольку одно только ее наличие сделает систему более громоздкой. Разработчики языка часто недооценивают этот момент. Я с готовностью допускаю, что временами было бы хорошо иметь некоторые возможности языка Ada, которые не имеют аналогов в Modula-2, но в то же время я спрашиваю, стоят ли они таких затрат. Цена значительна. Во-первых, хотя создание обоих языков началось в 1977 г., компиляторы языка Ada начали появляться только сейчас, в то время как языком Modula-2 мы уже пользуемся с 1979 г. Во-вторых, ходят слухи, что компиляторы Ada — это гигантские программы, состоящие из нескольких сотен тысяч строк команд, в то время как наш новейший компилятор Modula-2 измеряется лишь 5 тысячами строк. Признаюсь по секрету, что этот компилятор Modula-2 уже находится на пределе того уровня сложности, который еще можно понять, и я чувствую себя совершенно неспособным создать хороший компилятор для языка Ada. Но даже если пренебречь затратами труда по созданию излишне громоздких систем и стоимостью памяти для хранения их кода, то настоящие затраты скрыты в невидимых усилиях бесчисленных программистов, безуспешно пытающихся понять эти программы и эффективно их использовать.

Другой общей характеристикой упомянутых мною проектов был *выбор инструментария*. По моему мнению, инструментарий должен быть соизмерим с изделием; он должен быть по возможности прост, но не проще того. На деле выбор инструментария может привести к обратным результатам, когда большая часть всего проекта заключается в изготовлении такого инструментария. В проектах Euler, Algol-W и PL360 большое внимание уделялось разработке табличных методов восходящего синтаксического анализа. Позднее я снова вернулся к простому методу рекурсивного спуска "сверху вниз", который безусловно более понятен и при разумно выбранном синтаксисе достаточно мощен. При разработке аппаратного обеспечения Lilith мы пользовались лишь хорошим осциллографом, и только изредка у нас возникала необходимость в логическом анализаторе. Это было возможно благодаря относительно систематизированной, свободной от искусственных ухищрений концепции процессора.

Каждый отдельно взятый проект прежде всего является *обучающим экспериментом*. Лучше всего учиться, изобретая что-либо. Только непосредственно в *процессе разработки* проекта я смог в достаточной мере познакомиться с внутренне присущими ему трудностями и приобрести уверенность в том, что с деталями можно совладать. Я никогда не мог отделить друг от друга процессы проектирования и реализации языка, ибо стремление жестко определить язык, пренебрегая обратной связью с конструированием его компилятора, кажется мне самонадеянным и непрофессиональным. Поэтому я принимал участие в создании компилятора, и, как следствие, в микропрограммировании, в системном программировании на языках высокого уровня, занимался расчетом схем, компоновкой плат и даже скручиванием проводов. Это может показаться странным, но я люблю делать практическую работу собственными руками значительно больше, чем руководить группой. Кроме того, я понял, что исследователи более охотно признают руководителей, непосредственно входящих в рабочую группу, чем специалистов по организации труда, будь то менеджер или профессор университета. Я стараюсь не забывать, что обучение на хороших примерах часто является самым эффективным, а иногда и единственно возможным методом. И наконец, каждый из этих проектов был осуществлен благодаря энтузиазму и желанию преуспеть; все сотрудники знали, что они занимаются стоящим делом. Это, возможно, самое необходимое, но также и наиболее труднодостижимое условие успеха. Я был счастлив иметь сотрудников, которые позволили себе увлечься проектом, и я хочу воспользоваться этой возможностью, чтобы поблагодарить их всех за существенный вклад, который они внесли в нашу работу. Я признателен всем, кто так или иначе участвовал в разработках: непосредственно в нашей группе или помогая нам, тем, кто проверял наши результаты и сообщал о своих впечатлениях, подавал новые идеи, критикуя или одобряя нашу работу, тем, кто создавал сообщество пользователей. Без них ни Algol-W, ни Паскаль, ни Modula-2, ни Lilith не стали бы тем, чем они являются сейчас. Эта премия Тьюринга высоко оценивает также и их вклад. #

Об авторе. Никлаус Вирт (Niklaus K. Wirth) — профессор Швейцарского федерального технологического института (ETH) в Цюрихе, который Вирт закончил в 1958 г. и где получил специальность инженера в области электроники. Затем он продолжил свое обучение в Лавальском университете (Laval University) в Квебеке (Канада). В университете Калифорнии в Беркли (University of California at Berkeley, США) в 1963 г. Вирт защитил диссертацию. До 1967 г. работал доцентом на вновь образованном факультете компьютерных наук в Стенфордском университете (Stanford University, США), где он разработал язык PL360 и, в сотрудничестве с рабочей группой IFIP Working Group 2.1, язык Algol-W. В том же 1967 г. становится доцентом в университете Цюриха (University of Zurich), а в 1968 г. переходит в ETH, где в период с 1968 по 1970 годы разрабатывает язык Паскаль. Среди последующих проектов — разработка и реализация персонального компьютера Lilith, высокопроизводительной 16-разрядной рабочей станции с растровым дисплеем, создание языка Modula-2 (1978-1982 г.), и 32-разрядной рабочей станции Ceres (1984-1986 г.). Затем им были созданы языки Oberon и Oberon-2 (совместно с профессором Х.Мессенбоком), а также операционная система Oberon (1986-1989 г.). В 1984 г. профессор Вирт был удостоен почетной премии Алана Тьюринга (Turing Award), в 1989 г. — премии Max Petitpierre Prize, а также премии Science and Technology Prize от IBM Europe. Один из последних его проектов — система Lola System для разработки электронных схем (1995 г.).