Смиренный программист

В результате долгой цепочки случайностей первым весенним утром 1952 года я официально стал программистом, и насколько я могу судить, я был первым голландцем, выбравшим эту стезю. Помнится, самой забавной вещью была та неторопливость, с которой, по крайней мере в той части мира, где я жил, появлялась профессия программиста, неторопливость, в которую теперь даже поверить трудно.

После того, как я прозанимался программированием около трех лет, у меня состоялась беседа с А. Ван Вейнгаарденом, который был в то время моим боссом в Математическом центре Амстердама, беседа, за которую я буду благодарен ему до конца моих дней. Дело в том, что предполагалось, что я буду параллельно изучать теоретическую физику в Лейденском университете, и так как мне становилось все труднее и труднее совмещать два этих занятия, я должен был определиться: либо бросить программирование и стать почтенным физиком-теоретиком, либо кое-как доучиться физике до формального выпуска с минимальными усилиями, и затем стать..., а кем, кстати? Программистом? Но является ли это достойной профессией? В конце концов, что это такое - программирование? Где та солидная теоретическая основа, которая должна поддерживать его как уважаемую интеллектуальную дисциплину? Я довольно отчетливо помню, как я завидовал своим коллегам, которые работали с оборудованием: когда у них спрашивали об их профессиональных навыках, они по крайней мере могли сказать, что знают все о вакуумных лампах, усилителях и тому подобных вещах, тогда как я чувствовал, что, когда столкнусь с этим вопросом, мне останется только лишь развести руками. Полный опасений, я постучал в дверь Ван Вейнгаардена, спросив, могу ли я "поговорить с ним минутку"; несколько часов спустя я покинул его офис другим человеком. Внимательно выслушав мои проблемы, он согласился, что до сих пор наука программирования не так уж развита, но затем принялся неторопливо объяснять, что автоматические компьютеры - это надолго, что мы находимся в самом начале, и почему бы мне не стать одним из тех, кто призван сделать программирование уважаемой дисциплиной? Это стало поворотной точкой в моей жизни, и я формально закончил изучение физики так быстро, как только мог. Мораль этой истории - в том, что, конечно, мы должны быть очень осторожны, давая советы молодежи: иногда они следуют им!

Еще два года спустя, в 1957, я женился, и голландский обряд регистрации брака требовал указать профессию; я указал "программист". Но городские власти Амстердама не приняли документы на том основании, что такой профессии не существует. Хотите - верьте, хотите - нет, но в графе "профессия" моего свидетельства о браке значится забавная запись "физик-теоретик"!

Стоит ли говорить о том, как медленно профессия программиста пробивала себе путь в моей стране. С тех пор я немало повидал во всем мире, и мое общее впечатление таково, что в других странах, если не считать возможного сдвига дат, ее продвижение в целом было таким же.

Позвольте мне попытаться передать события этих давно минувших дней немного детальнее, чтобы попытаться получше понять, что же происходит сегодня. В процессе нашего анализа мы увидим, как много недоразумений относительно истинной природы программирования восходит к тем временам.

Первые автоматические электронные компьютеры были уникальными машинами, построенными в единственном экземпляре, и они находились в восхитительном окружении экспериментальной лаборатории. Как только дух автоматического компьютера в ней побывал, его воплощение становилось потрясающим вызовом электронной технологии того времени, и одно несомненно: мы не можем отказать в мужестве группам, которые решили попытаться построить это фантастическое сооружение. Потому что эти

сооружения действительно были фантастическими: оглядываясь назад, мы можем только изумляться тому, что эти первые машины вообще работали, по крайней мере, иногда. Самой обременительной проблемой было заставить машину работать и поддерживать ее в этом состоянии. Озабоченность физической стороной автоматических вычислений до сих пор находит отражение в названиях старейших научных сообществ этого направления, таких как the Association for Computing Machinery или the British Computer Society, - в названиях, в которых прослеживается явный намек на физическое оборудование.

А как же бедный программист? Что же, честно признаться, о нем вряд ли даже вспоминали. Во-первых, первые машины были столь громоздки, что вы вряд ли смогли бы сдвинуть их с места, и кроме того, они требовали такого объема работ по обслуживанию, что было вполне естественным, что их пытались использовать в той же лаборатории, где они и разрабатывались. Во-вторых, его практически невидимый труд был полностью лишен волшебного ореола: вы могли показать машину посетителям, и это было несравнимо зрелищнее, чем несколько листов, покрытых кодами. Но самое важное во всем этом - сам программист очень скромно оценивал собственный труд: вся значимость его работы определялась существованием этой чудесной машины. Так как эта машина была уникальной, он слишком хорошо сознавал, что его программы имеют исключительно местное значение. Также, поскольку было совершенно очевидно, что время жизни данной машины ограничено, то он знал, что весьма малая часть его работы будет иметь долговременное значение. Наконец, есть еще одно обстоятельство, которое оказало глубокое влияние на отношение программиста к своей работе: с одной стороны, помимо своей ненадежности, его машина обычно была слишком медленной и имела слишком мало памяти, так что он постоянно находился в прокрустовом ложе, тогда как с другой стороны ее причудливая система команд приводила к весьма неожиданным конструкциям. В те времена смышленый программист получал немалое интеллектуальное удовлетворение от изощренных трюков, посредством которых он умудрялся втиснуть невозможное в тесные рамки своего оборудования.

В те времена сложилось два мнения о программировании. Я упомяну о них сейчас и вернусь к ним позже. Одно мнение - действительно компетентный программист должен обладать парадоксальным мышлением и обожать заумные трюки; второе - программирование представляет собой не более чем оптимизацию эффективности вычислительного процесса в том или ином направлении.

Последнее мнение являлось результатом того, что весьма часто мощности имеющегося оборудования катастрофически не хватало, и в то время часто встречались наивные предположения, что как только более мощные машины станут доступны, программирование перестанет быть проблемой, ибо закончится борьба с тесными рамками оборудования, а ведь именно в этом и заключается суть программирования, не так ли? Но в следующие десятилетия произошло кое-что совершенно иное: стали доступны более мощные машины, не на один, а на несколько порядков величины мощнее прежних. Но вместо того, чтобы оказаться в состоянии бесконечного блаженства от того, что все проблемы программирования решены, мы оказались по самое горло в кризисе программирования! Как же это случилось?

Вот второстепенная причина: в одном-двух аспектах современные компьютеры значительно сложнее содержать, чем старые. Во-первых, мы получили прерывания ввода/вывода, происходящие в непредсказуемые и невоспроизводимые моменты времени; в сравнении со старыми последовательными машинами, которые прикидывались полностью детерминированными автоматами, это разительное изменение, и преждевременная седина многих системных программистов служит свидетельством тому, что нам не стоит легкомысленно отзываться о логических проблемах, порожденных этой возможностью. Во-вторых, мы получили машины, оборудованные многоуровневыми запоминающими устройствами, что ставит перед нами проблемы стратегии управления ею, которые, несмотря на обилие литературы по этому предмету, остаются весьма

скользкими. Слишком дорогая плата за возрастание сложности, обусловленное изменениями в структуре современных машин.

Но я назвал это второстепенной причиной; первостепенная же кроется в том, что... машины стали на несколько порядков мощнее! Говоря прямолинейно: пока машин не было вовсе, не существовало проблемы программирования; когда появились немногочисленные слабые компьютеры, программирование стало малозаметной проблемой; а теперь, когда у нас есть гигантские компьютеры, программирование само превратилось в гигантскую проблему. В этом смысле электронная промышленность не решила ни единой проблемы, она только породила их, создав проблему использования своей продукции. Другими словами, по мере того как мощность доступных машин выросла более чем в тысячу раз, стремление общества найти им применение выросло пропорционально, и бедный программист вынужден метаться буквально по минному полю. Возросшая мощь оборудования совместно с, возможно, еще более возросшей надежностью, сделали возможными решения, о которых программист даже не отваживался мечтать несколько лет назад. А теперь, спустя несколько лет, он должен мечтать о них и, более того, он обязан воплощать эти мечты в реальность! Удивительно ли, что мы оказались в кризисе программирования? Конечно же, нет, и как вы можете догадаться, это даже было предсказано заранее; но трагедия пророков, предсказывающих неприятности, состоит в том, что только лет через пять вы действительно осознаете, что они были правы.

Затем в середине шестидесятых годов произошло нечто ужасное: появились так называемые "компьютеры третьего поколения". Официальная литература утверждает, что соотношение "цена/производительность" было одной из главных целей разработки. Но если рассматривать как производительность рабочий цикл различных компонентов машины, вряд ли что-то сможет удержать вас от разработки, в которой большая часть производительности достигается посредством внутренних действий сомнительной необходимости. А если под ценой вы понимаете ту цену, которую приходится платить за оборудование, мало что удержит вас от разработки устройство, которое будет дьявольски трудно программировать: например, коды команд могут потребовать ранней компоновки либо от программиста, либо от системы, что представляет в действительности неразрешимый конфликт. И в немалой степени эти неприятные свойства стали реальностью.

Когда про эти машины было заявлено и стали известны их функциональные спецификации, многие из нас стали несчастны; по крайней мере, я стал. Можно было только ожидать, что подобные машины заполонят компьютерное сообщество, и поэтому важнее всего было разработать их как можно более надежными. Но в разработку вкрались столь серьезные изъяны, что я почувствовал, что одним ударом компьютерная наука отброшена как минимум на десятилетие: это была самая черная неделя в моей профессиональной жизни. Пожалуй, самое грустное сейчас - это то, что несмотря на все эти годы обескураживающего опыта, так много людей до сих пор искренне верят, что эти машины развиваются согласно законам природы. Они молчат о своих сомнениях, видя, как много подобных машин уже продано, и это внушает им ложное чувство безопасности, что в конце концов разработка не могла быть так уж плоха. Но при ближайшем рассмотрении эта линия защиты столь же убедительна, сколь и аргумент, что курение полезно для здоровья, потому что так много людей курят.

В связи с этим я сожалею, что не в обычаях научных журналов компьютерной тематики публиковать обзоры недавно анонсированных компьютеров, подобно обзорам научных публикаций: обозревать машины по меньшей мере столь же важно. И здесь я должен признаться: в начале шестидесятых я написал такой обзор, намереваясь отправить его в CACM. Но, несмотря на то, что немногие коллеги, которым я разослал текст на рецензию, побуждали меня сделать это, я все же не рискнул, опасаясь, что трудности либо для меня, либо для редакции окажутся слишком велики. За это малодушие я виню себя все

больше и больше. Трудности, которые я предвидел, были следствием общепринятых критериев, и хотя меня убедили в правильности выбранных мной критериев, я опасался, что мой обзор будет отвергнут как "выражение личного мнения". Я до сих пор считаю, что подобные обзоры были бы чрезвычайно полезны, и мне бы очень хотелось, чтобы они появились, потому что их принятие было бы безусловным признаком зрелости компьютерного сообщества.

Причина, по которой я выше уделил столь много внимания оборудованию, кроется в моей уверенности, что один из важнейших аспектов любого вычислительного инструмента - это его влияние на образ мыслей тех, кто пытается его использовать, а также в том, что у меня есть причины считать, что это влияние во много раз сильнее, чем принято думать. Давайте теперь переключим наше внимание на область программного обеспечения.

Здесь разнообразие было столь велико, что я вынужден ограничиться несколькими путевыми вехами. Я убежден в своей предвзятости, поэтому умоляю не делать никаких выводов и помнить о том, что я упоминаю здесь отнюдь не все достижения, которые высоко ценю.

Вначале был EDSAC в Кембридже, Англия, и я нахожу весьма впечатляющим, что с самого начала понятие библиотеки подпрограмм играло центральную роль при проектировании этой машины и способа ее использования. Прошло почти 25 лет, и обстановка в компьютерном мире радикально изменилась, но понятие базового программного обеспечения до сих пор с нами, и понятие замкнутой подпрограммы остается одной из ключевых концепций программирования. Мы должны признать замкнутые подпрограммы одним из величайших изобретений в программировании; оно пережило три поколения компьютеров и будет жить еще значительно дольше, поскольку представляет основу для реализации наших базовых механизмов абстракции. К огромному сожалению, их важность была недооценена при разработке компьютеров третьего поколения, в которых большое количество явно именуемых регистров арифметического устройства подразумевает большие накладные расходы для механизма подпрограмм. Но даже это не убило концепцию подпрограммы, и мы можем только молиться о том, чтобы эта мутация не оказалась наследуемой.

Вторым большим шагом вперед в области программирования, который мне хотелось бы упомянуть, является рождение FORTRAN'a. В то время это был проект безрассудной смелости, и люди, ответственные за него, заслуживают нашего величайшего восхищения. Было бы черной неблагодарностью винить их за недостатки, которые стали очевидны только после десятилетия интенсивного использования: группы, обладающие даром предвидения на десятилетие, весьма редки! Оглядываясь назад, мы должны оценить FORTRAN как успешную технологию кодирования, но очень мало помогающую мышлению, а эта помощь так нужна сейчас, что пришло время признать эту технологию устаревшей. Чем скорее мы сможем забыть, что FORTRAN когда-то существовал, тем лучше, потому что как ускоритель мысли он уже не адекватен: он растрачивает попусту наши мыслительные способности, он слишком рискован и, таким образом, слишком дорог, чтобы его использовать. Трагедия FORTRAN'a состоит в его общепринятости, приковывании умов тысяч и тысяч программистов к нашим прошлым ошибкам. Я ежедневно молюсь о том, чтобы больше моих коллег-программистов смогли найти пути к освобождению от проклятия совместимости.

Третий проект, который не хотелось бы оставить в забвении, - это LISP, очаровательный проект совершенно иной природы. Имея в своей основе так мало базовых принципов, он продемонстрировал замечательную стабильность. Кроме этого, LISP послужил основой для значительного числа в некотором роде наших наиболее изощренных компьютерных приложений. LISP был шутливо охарактеризован как "наиболее интеллектуальный способ использования компьютера не по назначению". Я считаю эту характеристику лестным комплиментом, поскольку она в полной мере

передает дух освобождения (разума): он помог многим нашим наиболее одаренным коллегам в обдумывании идей, доселе немыслимых.

Четвертый проект, о котором следует упомянуть, - это ALGOL 60. В то время как до сих пор программисты на FORTRAN'е продолжают осознавать свой язык в рамках конкретной реализации, с которой они работают - отсюда и избыток восьмеричных или шестнадцатеричных дампов, - в то время как определение языка LISP остается причудливой смесью описаний, что означает язык и как работает его механизм, знаменитый "Отчет об алгоритмическом языке ALGOL 60" - это продукт усилий по продвижению абстракции на один жизненно важный шаг дальше и по определению языка программирования способом, независимым от реализации. Кое-кто может возразить, что в этом отношении его авторы столь преуспели, что им удалось посеять серьезные сомнения, может ли он вообще быть реализован! "Отчет" великолепно демонстрирует мощь формального метода BNF, теперь справедливо известного как Backus-Naur-Form (Формализм Бэкуса-Наура, или Форма Бэкуса-Наура - прим. перев), и силу тщательно сформулированного английского языка, по крайней мере в изложении тех, кто владеет им столь же блестяще, как Питер Наур. Я думаю, справедливо будет заметить, что очень немногие документы, столь же короткие, как этот, оказали такое глубокое влияние на компьютерное сообщество. Легкость, с которой в последующие годы слова "ALGOL" и "ALGOL-подобный" использовались как незащищенные торговые марки, чтобы придать часть своей славы незрелым проектам, порой имеющим весьма слабое отношение к предмету, - это порой обескураживающий комплимент его основам. Сила BNF как определяющего механизма в ответе за то, что я расцениваю как одну из слабостей языка: переработанный и не слишком систематический синтаксис мог бы теперь быть втиснут всего в несколько страниц. Располагая столь мощным инструментом, каким является BNF, "Отчет об алгоритмическом языке ALGOL 60" должен был бы быть значительно короче. Кроме того, я весьма усомнился в механизме параметров ALGOL'а 60: он предоставляет программисту столько комбинаторной свободы, что его надежное использование требует от программиста строгой дисциплины. Помимо того, что он дорог в реализации, он кажется также опасным в использовании.

Наконец, хотя этот предмет не из приятных, я должен упомянуть PL/1, язык программирования, документация которого обладает устрашающими размерами и сложностью. Использование PL/1 больше всего напоминает полет на самолете с 7000 кнопок, переключателей и рычагов в кабине. Я совершенно не представляю себе, как мы можем удерживать растущие программы в голове, когда из-за своей полнейшей вычурности язык программирования - наш основной инструмент, не так ли! - ускользает из-под контроля нашего интеллекта. И если мне понадобится описать влияние, которое PL/1 может оказывать на своих пользователей, ближайшее сравнение, которое приходит мне в голову, - это наркотик. Я помню лекцию в защиту PL/1, прочитанную на симпозиуме по языкам программирования высокого уровня человеком, который представился одним из его преданных пользователей. Но после похвал в адрес PL/1 в течение часа он умудрился попросить добавить к нему около пятидесяти новых "возможностей", не предполагая, что главный источник его проблем кроется в том, что в нем уже и так слишком уж много "возможностей". Выступающий продемонстрировал все неутешительные признаки пагубной привычки, сводящейся к тому, что он впал в состояние умственного застоя и может теперь только просить еще, еще, еще... Если FORTRAN называют детским расстройством, то PL/1, с его тенденциями роста подобно опасной опухоли, может оказаться смертельной болезнью.

Я немало рассказал о прошлом. Но ведь нет никакого смысла делать ошибки, если мы впоследствии не способны извлечь из них урок. В действительности я считаю, что мы научились столь многому, что через несколько лет программирование станет видом деятельности, в корне отличным от того, каким оно было до сих пор, настолько отличным, что нам, пожалуй, лучше приготовиться к шоку. Позвольте мне набросать для вас одну из

возможных картин будущего. На первый взгляд, такая точка зрения на программирование в, возможно, уже ближайшем будущем может поразить вас своей крайней фантастичностью. Поэтому позвольте мне привести доводы, из которых можно сделать заключение, что она может быть весьма реалистичной.

Эта точка зрения такова: еще до конца семидесятых мы сможем разрабатывать и реализовывать такие виды систем, которые нынче едва по силам нашим возможностям как программистов, ценой всего лишь несколько процентов тех человеко-лет, в которые они нам обходятся сегодня, и кроме того, эти системы будут практически свободны от ошибок. Эти два усовершенствования идут рука об руку. В последнем отношении программное обеспечение кажется отличным от множества других продуктов, для которых, как правило, более высокое качество подразумевает более высокую цену. Те, кто желает получить действительно надежное программное обеспечение, обнаружат, что они должны искать средства избежать большинство ошибок с самого начала, и в результате сам процесс программирования станет дешевле. Если вам нужны более эффективные программисты, вы обнаружите, что они не должны растрачивать свое время на отладку, - они с самого начала не должны вносить ошибки в программы. Другими словами: обе цели требуют одной и той же перемены.

Такая значительная перемена в столь короткий период времени была бы революцией, и для всех, кто строит свои надежды на будущее на осторожной экстраполяции недавнего прошлого, основываясь на неписанных законах социальной и культурной инертности, вероятность, что такая перемена произойдет, должна казаться пренебрежимо малой. Но мы знаем, что революции все же происходят время от времени! И какие же изменения она принесет нам?

Похоже, должны выполниться три основных условия. Во-первых, во всем мире должны признать необходимость перемен; во-вторых, для них должны быть достаточно сильны экономические потребности; и в-третьих, перемены должны быть осуществимы технически. Давайте обсудим эти три условия в этом же порядке.

В отношении признания необходимости более высокой надежности программного обеспечения, я надеюсь, больше нет возражений. Всего несколько лет назад это было не так: разговоры о кризисе программного обеспечения расценивались как кощунство. Поворотной точкой стала "Конференция по инженерии программного обеспечения" в октябре 1968 г. в Гармише, конференция, которая породила сенсацию, ибо именно на ней впервые открыто было признано существование кризиса программного обеспечения. К настоящему моменту стало общепризнанным, что разработка любой крупной и сложной системы будет очень трудной задачей, и каждый раз, встречаясь с людьми, ответственными за такое мероприятие, можно заметить их озабоченность проблемой надежности, и это действительно так. Короче говоря, кажется, наше первое условие выполнено.

Теперь об экономической потребности. Теперь часто встречается мнение, что в шестидесятые программирование было чересчур высокооплачиваемой профессией, и в ближайшие годы зарплаты программистов, вероятно, будут понижаться. Обычно это мнение высказывается в связи с экономическим спадом, но это могло бы быть и симптомом кое-чего другого, и весьма резонным, а именно - возможно, программисты истекшего десятилетия выполнили свою работу вовсе не так хорошо, как следовало бы. Общество разочаровывается в производительности труда программистов и в результатах их усилий. Но есть и другой фактор, куда более весомый. В сложившейся ситуации вполне естественно, что для заданной системы стоимость разработки программного обеспечения имеет примерно тот же порядок, что и стоимость необходимого оборудования, и общество более-менее смирилось с этим. Но производители оборудования уверяют нас, что в следующем десятилетии ожидается снижение стоимости оборудования раз в десять. Если разработка программного обеспечения будет и далее столь же дорогим и неуклюжим процессом, каким она является сейчас, баланс будет

окончательно нарушен. Нельзя надеяться, что общество смирится с этим, и таким образом мы должны научиться программировать на порядок эффективнее. Другими словами, пока на долю оборудования приходилась наиболее весомая часть бюджета, программированию сходили с рук неуклюжие технологии, но долго в тени этого зонтика прятаться не поучится. Короче говоря, наше второе условие также выполнено.

А теперь - третье условие: выполнимо ли все это технически? Я думаю, могло бы быть выполнимо, и приведу шесть аргументов в поддержку этого мнения.

Изучение структуры программ выявило, что программы - даже альтернативные варианты, которые решают те же самые задачи и оперируют теми же математическими константами - могут разительно отличаться по своей интеллектуальной управляемости. Было открыто несколько правил, нарушение которых либо существенно ослабляет, либо вовсе разрушает интеллектуальную управляемость программы. Эти правила делятся на два вида. Правила первого вида накладываются автоматически выбором адекватного языка программирования. Примеры - исключения операторов goto и процедур с более чем одним выходным параметром. Для правил второго вида я по меньшей мере - возможно, из-за недостаточной компетентности с моей стороны - не вижу способа механического применения, потому что, мне кажется, для этого потребуется некий автомат для доказательства теорем, для которого я не располагаю доказательством существования. Поэтому сейчас (и возможно, навсегда) правила второго вида представляют собой элементы дисциплины, требуемой от программиста. Некоторые из этих правил, которые я держу в голове, настолько ясны, что им можно обучить и что никогда не потребуется спорить, нарушает ли их данная программа или нет. Примеры: требования к циклам, которые не должны записываться без приведения доказательства их завершения, а также без установки соотношений, инвариантность которых не нарушается при циклическом выполнении оператора.

Я предлагаю отныне строго придерживаться разработки и реализации только интеллектуально-управляемых программ. Если кто-то опасается, что это ограничение столь строго, что мы не в состоянии жить с ним, я могу его уверить: класс интеллектуально-управляемых программ все еще остается достаточно обширным, чтобы в нем содержались многие весьма реалистичные программы для любых задач, имеющих алгоритмическое решение. Не следует забывать, что производство программ - это не наше дело, наше дело - разработка классов вычислений, которые демонстрируют желаемое поведение. Предложение ограничиться интеллектуально управляемыми программами - основа первых двух из шести объявленных аргументов.

Первый аргумент: так как программисту необходимо рассматривать только интеллектуально-управляемые программы, с выбором альтернатив гораздо легче справиться. Второй аргумент: как только мы решили ограничиться подмножеством интеллектуально-управляемых программ, мы раз и навсегда достигли значительного сжатия пространства рассматриваемых решений. И этот аргумент отличен от первого.

Третий аргумент основывается на конструктивном подходе к проблеме корректности программ. Сегодня обычная технология - сделать программу, а затем оттестировать ее. Но тестирование программы может весьма эффективно продемонстрировать наличие ошибок, но безнадежно неадекватно для демонстрации их отсутствия. Единственный эффективный способ значительно поднять уровень доверия к программам - предоставить убедительное доказательство их корректности. Но не следует сначала писать программу, а затем доказывать ее корректность, потому что тогда требование предоставить доказательство только увеличит обузу бедного программиста. Наоборот, программист должен позволять расти доказательству корректности и программе совместно. Третий аргумент в основном основывается на следующем наблюдении. Если кто-то сначала спросит себя, какова будет структура убедительного доказательства и, найдя ее, затем построит программу, удовлетворяющую требованиям этого доказательства, тогда эти заботы о корректности обернутся весьма эффективным

эвристическим указателем пути. По определению этот подход применим только в случае, если мы ограничиваемся интеллектуально-управляемыми программами, но он дает нам эффективные средства для поиска удовлетворительного решения среди них.

Четвертый аргумент относится к образу, которым количество интеллектуальных усилий, необходимых для разработки программы, зависит от длины программы. Было высказано предположение, что имеется некий закон природы, согласно которому количество необходимых интеллектуальных усилий растет как квадрат длины программы. Но, слава богу, никто не смог доказать справедливость этого закона. А не смог потому, что это не должно быть правдой. Все мы знаем, что единственный умственный инструмент, посредством которого весьма ограниченный разум может охватить мириады различных вариантов, называется "абстракцией"; в результате эффективное использование мощи абстракции должно расцениваться как одно из наиболее жизненно важных действий компетентного программиста. В этой связи было бы полезно упомянуть, что цель абстракции - не напустить тумана, а создать новый семантический уровень, на котором она будет абсолютно точна. Разумеется, я попытался найти фундаментальную причину, которая помешала бы нашим абстрактным механизмам быть достаточно эффективными. Но как бы я ни старался, я не мог найти эту причину. В результате я склоняюсь к мнению - до сих пор не опровергнутому практикой - что при надлежащем применении наших способностей к абстракции интеллектуальные усилия, необходимые для постижения или понимания программы, должны расти не более чем пропорционально длине программы. Но побочный продукт этих исследований может иметь еще большее практическое значение и фактически является основой моего четвертого аргумента. Этим побочным продуктом является признание нескольких шаблонов абстракции, которые играют жизненно-важную роль во всем процессе составления программ. Сейчас о таких шаблонах абстракции известно достаточно много для того, чтобы посвятить каждому из них отдельную лекцию. То, сколько знаний и навыков хранят в себе эти шаблоны абстракции, поразило меня, когда я осознал, что если бы они были общеизвестны пятнадцать лет назад, переход от BNF к синтаксически-управляемым компиляторам, например, занял бы несколько минут вместо нескольких лет. Поэтому я выдвигаю наши последние познания в области жизненно-важных абстракций как четвертый аргумент.

Теперь переходим к пятому аргументу. Он относится к влиянию инструментов, которые мы пытаемся использовать, на наш образ мышления. Я наблюдаю культурную традицию, которая, по всей вероятности, уходит корнями в эпоху Возрождения: игнорировать это влияние и рассматривать человеческую мысль как первичную и главенствующую над тем, что ей создано. Но когда я начинаю анализировать свои собственные мыслительные привычки и привычки моих друзей, я прихожу, нравится мне это или нет, к совершенно иному заключению, а именно: инструменты, которые мы пытаемся использовать, и язык и обозначения, которые мы используем для выражения или записи наших мыслей, являются основным фактором, который определяет, о чем мы вообще можем мыслить и что можем выразить! Анализ влияния, которое язык программирования оказывает на своих пользователей, и признание факта, что к настоящему времени мощь нашего мозга является наиболее скудным ресурсом, совместно дают нам новый набор эталонов для сравнения относительных достоинств различных языков программирования. Компетентный программист хорошо знает о том, что объем его черепной коробки крайне ограничен; поэтому он подходит к задаче программирования с предельным смирением, и помимо прочего он избегает заумных трюков как чумы. В случае хорошо известных диалоговых языков мне часто толковали со всех сторон, что как только сообщество программистов обзавелось терминалами, появился специфический феномен, который даже получил широко известное название - "однострочники". Он принимает одну из двух форм: один программист кладет перед другим однострочную программу и либо гордо рассказывает, что она делает, а затем спрашивает: "А ты можешь закодировать это меньшим количеством символов?" - как будто это имеет какую-то

практическую ценность, - либо просто спрашивает: "Угадай, как это работает!". Из этого наблюдения мы можем заключить, что в качестве инструмента этот язык является открытым вызовом для хитроумных трюков; и пока именно это может быть объяснением его привлекательности для некоторых, которым нравится демонстрировать свою гениальность, простите меня, но я вынужден рассматривать его как самую предосудительную вещь, насколько это вообще может относиться к языку программирования. Еще один урок, который мы должны извлечь из недавнего прошлого, - это то, что разработка "более богатых возможностями" или "более мощных" языков программирования была ошибкой в том смысле, что эта причудливая монструозность, это нагромождение идиосинкразий в действительности неуправляемы как механически, так и ментально. Я предвижу великое будущее для очень систематических и очень умеренных языков. Говоря об умеренности, я имею в виду, к примеру, что не только выражение "for" из ALGOL 60, но и цикл "DO" из FORTRAN'a могут оказаться выкинутыми за свою причудливость. Я провел небольшой программный эксперимент с действительно опытными добровольцами, подсунув им что-то достаточно неожиданное и непредвиденное. Ни один из моих добровольцев не нашел очевидного и самого элегантного решения. При более внимательном рассмотрении оказалось, что это имеет общий для всех источник: их представление о повторении было так тесно связано с идеей связанной управляющей переменной, что их рассудок был слишком зашорен, чтобы заметить очевидное. Их решения были менее эффективны, беспричинно слишком трудны для понимания, и им потребовалось слишком много времени на их поиск. Это был весьма поучительный, но в то же время обескураживающий опыт. Наконец, можно надеяться, что языки программирования завтрашнего дня будут значительно отличаться от тех, к которым мы привыкли сегодня: в значительно большей степени, чем до сих пор, они будут подталкивать нас к отражению в структуре того, что мы пишем, всех абстракций, необходимых для концептуального охвата сложности предмета нашей разработки. Это все, что касается большей адекватности наших будущих инструментов, что лежит в основе пятого аргумента.

К слову, мне хотелось бы вставить предупреждение для тех, кто отождествляет сложность задачи программирования с борьбой против неадекватности наших нынешних инструментов, потому что они могли бы заключить, что, как только наши инструменты станут более адекватными, программирование перестанет быть проблемой. Программирование останется весьма трудным, потому что как только мы избавимся от обременительных деталей, мы окажемся свободны для того, чтобы приняться за задачи, которые лежат пока за пределами наших возможностей.

Вы можете оспаривать мой шестой аргумент, потому что не так-то легко собрать экспериментальные свидетельства в его поддержку, но это не изменит моей веры в его истинность. До сих пор я не упоминал слово "иерархия", но я считаю, что справедливо сказать, что это ключевая концепция всех систем, основанных на хорошо структурированном решении. Я мог бы даже пойти на шаг дальше и написать статью в подтверждение этого, а именно: мы можем в действительности удовлетворительно решить только те задачи, которые окончательно допускают хорошо структурированное решение. На первый взгляд, такая точка зрения на свойственные человеку ограничения может поразить вас своим пессимизмом по поводу свойственных нам затруднений, но я так не считаю, совсем наоборот! Лучший способ научиться жить, несмотря на наши ограничения, - это знать их. К тому времени, когда мы станем достаточно скромны, чтобы пытаться находить только структурированные решения, потому что другие усилия нам попросту не по зубам, мы будем делать все, что в наших силах, чтобы избежать всех интерфейсов, идущих во вред нашей способности структурировать систему подходящим образом. И я могу только надеяться, что это будет раз за разом приводить к открытию, что изначально неподдающаяся задача в конце концов может быть структурирована. Любой, кто видел, как большинство проблем фазы компиляции, называемой "генерацией кода", может быть

локализовано благодаря забавным особенностям машинного кода, знает простой пример того, что я имею в виду. Широкое применение хорошо структурированных решений - мой шестой и последний аргумент в пользу технической возможности революции, которая может разразиться в текущем десятилетии.

В принципе, я оставляю на ваше усмотрение решать для себя, насколько серьезно отнестись к моим доводам, хорошо зная, что я никого не могу заставить насильно разделить мои надежды. Как и каждая серьезная революция, она вызовет серьезное сопротивление, и каждый может спросить себя, где ожидать появления консервативных сил, противодействующих такому развитию. Я не ожидаю их появления в большом бизнесе и даже в компьютерном бизнесе; я скорее ожидаю их в образовательных учреждениях, которые сегодня занимаются обучением, и в консервативных группах пользователей компьютеров, которые считают свои старые программы столь важными, что не находят целесообразным переписать и улучшить их. В этой связи грустно наблюдать, как в некоторых университетских городках выбор центрального компьютера слишком часто определяется потребностями немногих, но дорогих приложений, игнорируя вопрос, сколько тысяч "малых пользователей", желающих писать собственные программы, пострадают от этого выбора. Например, слишком часто кажется, что ученые, занимающиеся физикой высоких энергий, шантажируют научное сообщество стоимостью экспериментального оборудования. Самый простой ответ, конечно, - это категорическое отрицание технической осуществимости, но боюсь, что вам понадобятся весьма состоятельные доводы для этого. Увы, замечание, что интеллектуальный потолок нынешнего среднего программиста не даст революции свершиться, не внушает уверенности: когда другие начнут программировать более эффективно, он непременно выйдет за эти рамки.

Могут быть также препятствия политического характера. Если даже мы знаем, как учить завтрашних профессиональных программистов, неочевидно, что общество, в котором мы живем, позволит нам делать это. Первый эффект от обучения методологии - а скорее, распространения знаний - состоит в том, что, расширяя возможности уже возможного в данный момент, тем самым усиливается различие в интеллектуальных способностях. В обществе, где система образования используется как инструмент для создания однородной культуры, в котором сливкам не дают подниматься к вершине, обучение компетентных программистов может оказаться политически некорректным.

Позвольте мне перейти к заключению. Автоматические компьютеры были с нами уже четверть века. Они оказали огромное влияние на наше общество как могучие инструменты, но при всей этой мощности их влияние будет только рябью на поверхности нашей культуры по сравнению с гораздо более глубоким влиянием, которое они окажут своей возможностью бросить беспрецедентный интеллектуальный вызов за всю культурную историю человечества. Похоже, что иерархические системы обладают тем свойством, что нечто, рассматриваемое как неделимое целое на одном уровне, рассматривается как составной объект на следующем, более низком уровне с большей детализацией; в результате дискретность пространства или времени, применимая на каждом уровне, уменьшается а порядок величины при переходе от одного уровня к другому, следующему за ним более низкому. Мы воспринимаем стену через понятие кирпичей, кирпичи - через понятие кристаллов, кристаллы - через понятие молекул и так далее. В результате количество уровней, которые могут быть осмысленно выделены в иерархической системе, в некотором роде пропорционально логарифму отношения между наибольшим и наименьшим дискретами, и поэтому, если только это соотношение не чрезмерно велико, мы можем ожидать появления не слишком большого числа уровней. В области компьютерного программирования наш базовый строительный блок имеет дискретность времени менее микросекунды, но наша программа может потребовать несколько часов вычислений. Я не знаю никакой другой технологии, перекрывающей отношение 1010 и более: компьютер, благодаря его фантастической скорости, кажется,

впервые предоставил нам среду, в которой артефакты с высокой степенью иерархичности одновременно и возможны, и необходимы. Этот вызов, а именно противостояние задаче программирования, столь уникален, что новый опыт может рассказать нам очень много нового о нас самих. Он может углубить наше понимание процессов разработки и созидания, он может дать нам лучший контроль над организацией нашего мышления. Если бы он не сделал этого, на мой взгляд, мы вообще не заслуживаем компьютеров!

Он уже преподал нам несколько уроков, и один из них, который я выбрал, чтобы акцентировать внимание в этом докладе, заключается в следующем. Мы будем программировать гораздо лучше, если только подойдем к задаче, полностью оценивая ее потрясающую сложность, если только мы будем придерживаться скромных и элегантных языков программирования, если только мы примем во внимание свойственные человеческому разуму ограничения и подойдем к задаче как Очень Смиренные Программисты.