Карло Пешио

Никлаус Вирт о культуре разработки ПО

Источник: Открытые системы, #01/1998.

"Никлаус Вирт (Niklaus Wirth) бесспорно является одним из наиболее известных и почитаемых мыслителей в мире информатики. Профессор ETH Institute в Цюрихе, Швейцария, он является автором новаторских языков и систем программирования Pascal, Modula 2 и Oberon. В начале 70-х гг. он был одним из тех, кто ввел в практику принцип пошаговой разработки ПО. Он автор многих известных книг, среди которых признанные классическими "Algorithms + Data Structures = Programs" и "Systematic Programming". Н. Вирт известен определенными и выражаемыми с абсолютной точностью взглядами на современное состояние культуры разработки ПО, поэтому интервью с ним всегда вызывают оживленную реакцию в компьютерном сообществе.

К.П.: Вы один из влиятельнейших ученых, причем преуспели как в теории, так и в практике разработки ПО. Часто, однако, университетская среда и "реальный мир" воспринимаются как едва пересекающиеся пространства. Когда я разговариваю с профессором, а потом о том же — с программистом, то нередко отмечаю, как по-разному они думают о разработке ПО. Недавно выпущенный IEEE обзор о перспективах развития процессов разработки ПО показал, сколь сильно отличаются мнения, высказываемые учеными и практиками. Как один из немногих, к кому прислушиваются в обоих этих мирах, поделитесь своими взглядами на этот предмет (или — возможно — секретами?)

Н.В.: Если и есть какой секрет, так только в том, как удается быть одновременно и программистом, и профессором. Вообще же именно та прискорбная и ненормальная ситуация, когда между практиками и теми, кто их учит, возникла стена непонимания — и есть источник многих проблем. Впрочем, современные профессора не столь уж много времени проводят в учебных аудиториях, не говоря уже о собственно разработке ПО: они "определяют политику", формулируют предложения, добывают финансовую поддержку, консультируют, путешествуют, дают интервью и т.п. И в результате теряют контакт со столь быстро меняющимся предметом. Они теряют способность проектировать, вообще ухватывать суть дела. И потому вынуждены мигрировать в пространство интеллектуальных головоломок, представляющих лишь академический интерес. Тому же и учат. Для меня изобретение нового языка программирования никогда не было самоцелью. Просто в этом возникала чисто практическая необходимость, а имевшиеся языки были неудовлетворительны. Например, Modula и Oberon были побочными продуктами работы над рабочими станциями Lilith (1979) и Ceres (1986). Ну а то, что я одновременно был университетским профессором, повлияло на проектируемые языки и системы в том отношении, что я старался сделать их максимально простыми; поэтому, используя их в учебном процессе, я мог концентрироваться на сущностных аспектах программирования, а не на деталях языка и нотации. Безусловно, стена между наукой и практикой ненормальна.

К.П.: Вы, конечно, знаете о понятии "достаточно хорошего ПО" (good-enough software), которое популяризирует Эд Йордон (Ed Yourdon). Во многом, это просто рационализация того, что есть современная реальность в мире программной индустрии: компания, которая первой выбрасывает на рынок не вполне совершенный продукт с богатой функциональностью, с большой вероятностью выйдет победителем в борьбе с конкурентом, стремящимся произвести качественный продукт. Можно ли здесь что-то предпринять? Я полагаю, многие разработчики были бы счастливы создавать более качественные продукты, но в то же время они вынуждены спешить во имя корпоративного выживания. Можно, конечно, мечтать, что пользователи будут ориентироваться на действительно качественный товар, но вряд ли это реально.

Н.В.: "Достаточно хорошее ПО" редко бывает действительно достаточно хорошим — это грустное проявление духа нынешнего времени, в котором исчезает личная гордость за свою работу. Сама мысль о том, что человек может испытывать удовлетворение от хорошо выполненной работы — просто потому, что эта работа творческая и профессиональная, признана абсурдной. Не ценится ничего, кроме экономического успеха и его денежного выражения. Отсюда можно сделать вывод: наша профессия сделалась просто тривиальной работой. Но по моему убеждению, нельзя ожидать качественно выполненной работы, если не отдавать ей себя полностью, если нет личного удовлетворения, более того — удовольствия от нее. В нашей профессии точность и совершенство — это не роскошь, а простая необходимость.

К.П.: Как Вы знаете, идут оживленные споры о специфике software engineering как профессии. Фактически, многие разработчики — практики не получили надлежащего образования и даже какого-либо значительного опыта. Должны ли инженеры-программисты обязательно быть дипломированными специалистами — подобно инженерам в других отраслях? Нужно ли что-то менять в программах обучения с целью повышения их эффективности? Каково, на Ваш взгляд, идеальное образование для инженера-программиста?

Н.В.: Недавно я познакомился с финальным отчетом исследовательского проекта, профинансированного Швейцарским Федеральным Научным Фондом. Среди весьма наивных вопросов, на которые этот проект должен был дать ответ, были и такие: во-первых, как легко можно освоить программирование, особенно не имея специального образования? А во-вторых, как могут быть реализованы такие механизмы, которые могут "скрыть" трудные аспекты параллельного программирования? Современному программированию более тридцати лет, и мы уже вполне убедились, что проектирование и разработка сложного ПО является по самой своей природе нетривиальной и нелегкой деятельностью. Это факт, и он не меняется от того, что индустрия десятилетиями привлекала неофитов занять свободные программистские вакансии утверждением, что программировать — легко. Позднее, когда сомнения начали посещать даже таких рекламодателей, они нашли выход в обещаниях обеспечить широкий набор инструментальных средств, способных облегчить решение трудных программистских задач. Инструменты получили самодовлеющий статус; получило распространение представление, что правильно выбранный инструментарий в сочетании с хитроумными приемами и серьезными методами менеджмента, способны творить чудеса. Тогда Эдсгар Дейкстра (Edsger Dijkstra) назвал software engineering как "искусство программирования без умения это делать".

Н.В.: На самом деле, беды software engineering происходят вовсе не от отсутствия инструментов или хорошего менеджмента, а от недостатка технической компетентности. Хороший проектировщик должен опираться на опыт, на строгое логическое мышление; и на педантичную точность. Никакая чудесная магия помочь не может. В свете всего этого особенно грустно, что во многих университетских программах по информатике "программированием в большом" (programming in the large) пренебрегают. Проектирование не заняло надлежащего места в программах по подготовке специалистов. Как результат, software engineering превратилось в Эльдорадо для хакеров. Программировать без царя в голове стало условием профессионального выживания: чем более хаотичной выглядит программа, тем меньше опасность, что кто-то возьмет на себя труд проинспектировать этот код и развенчать как саму программу, так и ее автора.

К.П.: Еще об обучении. Многие полагают, что легче изучать объектно-ориентированный подход будучи не отягощенным предыдущим опытом с использованием других парадигм программирования. Я с этим абсолютно не согласен и думаю, что профессиональный разработчик должен быть знаком со многими парадигмами. По моему мнению, ранние работы по Software Engineering, такие как пионерские статьи Дэвида Парнаса (David Parnas) или Ваша классическая книга по программированию "Systematic Programming" столь же полезны сегодня, как и 20 лет назад. Что Вы думаете об этом?

Н.В.: Многие люди относятся к стилям и языкам программирования как к религиозным конфессиям: если вы принадлежите к одной из них, то не можете принадлежать к другой. Но это ложная аналогия, и она сознательно поддерживается по причинам коммерческого порядка. Объектно-ориентированное программирование вышло из принципов и понятий традиционного процедурного программирования. Скажу больше: в ООП не добавлено ни одного действительно нового понятия; просто по сравнению с процедурным оно делает значительно более сильный акцент на двух понятиях. Первое — это привязка процедуры к составной переменной, что и послужило оправданием для введения терминов "объект" и "метод". Средством для такой привязки является процедурная переменная (или поле записи — record field), доступная в языках программирования с середины 70-х гг. Второе понятие — это конструирование нового типа данных (названного "подкласс") путем расширения заданного типа ("суперкласс").

Н.В.: Стоит заметить, что вместе с ООП пришла совершенно новая терминология, имевшая целью затемнить происхождение его корней. Таким образом, если раньше вы могли инициировать активность процедуры путем ее вызова, то теперь должны посылать сообщение методу. Новый тип строится не расширением заданного, а определением подкласса, который наследует от суперкласса. Это вообще интересный феномен, когда многие люди узнают о таких важных (и древних!) понятиях, как тип данных, инкапсуляция и (возможно) скрытие информации, лишь когда начинают изучать объектно-ориентированное программирование. Что ж, одно это оправдывает излишний шум вокруг ООП, даже если позднее эти неофиты ничего этого и не используют.

Н.В.: Тем не менее, я склонен рассматривать ООП как аспект более общего понятия "программирования в большом" (programming in the large) — тот аспект, что логически следует за "программированием в малом" (programming in the small) и уже поэтому требует надлежащего знания процедурного программирования. Статическая модуляризация — это первый шаг навстречу ООП; этот аспект намного легче понять и освоить, чем полное ООП, к тому же в большинстве случаев этого достаточно для написания хороших программ. Вот почему очень жаль, что этим аспектам в большинстве языков пренебрегли (за исключением Ada).

Н.В.: Я бы не сказал, что распространившаяся практика ООП реализовала все свои потенции. Наша конечная цель — расширяемое программирование (extensible programming). Под этим я понимаю возможность конструирования таких иерархий модулей, когда каждый модуль добавляет новую функциональность в систему. Расширяемое программирование подразумевает, что добавление модуля возможно без необходимости вносить какие-либо изменения в существующие модули — не должно быть необходимости даже их перекомпилировать. Новые модули не только добавляют новые процедуры, но — что более важно — добавляют также новые (расширенные) типы данных. Мы продемонстрировали практичность и экономичность этого подхода при проектировании Oberon System.

К.П.: Недавно мне попал в руки рекламный проспект на Delphi фирмы Borland (которое, как Вы знаете, является некой разновидностью объектно-ориентированного Pascal с расширением для управления событиями), в котором говорилось: "Delphi предоставляет разработчику язык, почти столь же удобочитаемый как BASIC...". Очевидно, это была цитата из описания продукта. Однако для меня это звучит абсурдно. "Почти столь же удобочитаем", как и язык без надлежащим образом определенного понятия типа данных?! С другой стороны, мы не может пренебречь тем фактом, что BASIC (Visual Basic) оказался победителем в конкурентной борьбе и является, вероятно, первым примером коммерческого языка на громадном рынке компонентов. Что Вы думаете об этом, как отец Pascal? На самом ли деле BASIC победил? И если да, то почему?

Н.В.: Мы должно осторожно употреблять такие термины, как "читаемость", "дружественность к пользователю" и им подобные. В лучшем случае, они туманны и часто ссылаются на такие плохо определенные сущности, как вкусы и установившиеся привычки. Но то, что общепринято, отнюдь не обязательно действительно так уж удобно. В контексте языков программирования, возможно "удобочитаемый" (readable) следует заменить на "подходящий для формальных умозаключений" (formal reasoning). Например, математические формулы едва ли могут удостоиться похвал как легко прочитываемые, но они позволяют выполнять формальный вывод свойств, которые в принципе не могут быть получены из туманных, нечетких, неформальных, "дружественных к пользователю" описаний.

Н.В.: Возьмем для примера конструкцию WHILE B DO S END. Она имеет то замечательное свойство, что вы можете вполне быть уверены в том, что B будет иметь значение "false" после выполнения этого оператора — независимо от S. И если вы находите свойство P, которое является инвариантным по отношению к S, вы можете предположить, что P останется таким же по завершении. Именно такой тип логического вывода помогает в надежном создании программ и весьма радикально сокращает время, затрачиваемое на тестирование и отладку. Хорошие языки не только строятся на основе математических понятий, делающих возможными логические умозаключения относительно программ; они еще поддерживают небольшое число понятий и правил, которые могут свободно комбинироваться друг с другом. Если определение языка требует руководства в 100 и более страниц, и если это определение содержит ссылки на механическую модель исполнения (например, компьютер), то это верный признак неадекватности этого языка. Увы, в этом отношении, Algol 1960-го г. рождения далеко впереди большинства своих преемников, в том числе и тех, что в фаворе сегодня.

К.П.: Еще один очень популярный язык сегодня — C++. Я знаю, что Вы не испытываете особого восторга от него, и во многих случаях лучше бы использовать более безопасный язык. Однако, не будет ли мудрее помочь программистам вместо того, чтобы бороться с ними. Например, во многих случаях, C++ - программисты были бы рады использовать более безопасную версию этого языка, тем более, что далеко не все связаны необходимостью поддерживать 100%-совместимость с языком Си. Версия C++, в которой указатели и массивы были бы ясно разделены, и где можно было бы получать предупреждающее сообщение, когда, к примеру, вы присваиваете значение с плавающей точкой переменной целого типа и т.д., могла бы помочь в создании лучших программ — и без необходимости изучать совершенно новый язык. Я понимаю, что много более приятно спроектировать концептуально чистый язык, чем попытаться сделать может быть не идеальный, но широко используемый более безопасным: но ведь если этот превосходный "чистый" язык используется только ограниченным кругом людей, то продвигаем ли мы на самом деле вперед состояние дел в области разработки программного обеспечения?

Н.В.: Моя позиция в этом вопросе не предусматривает компромиссов. Как университетский профессор, я обучаю будущих программистов. Пытаясь делать это на максимально высоком уровне, я должен подавать фундаментальные понятия программирования столь ясно, четко и кратко, сколь это возможно. И я определенно не позволю, чтобы неадекватная нотация помешала мне в этом. Если студенты восприняли важнейшие идеи, приобрели разносторонний опыт и знания, то у них не будет больших проблем в адаптации других языков (хотя они обычно не испытывают положительных эмоций, сталкиваясь с неадекватностью новых языков). Хотя, конечно, интересно, почему никто из заинтересованных лиц и организаций в столь большом сегменте программной индустрии до сих пор по настоящему не озаботился решением той задачи, о которой Вы сказали: определить безопасное подмножество C++. Я вижу здесь две причины. Во-первых, программистский мир все время жаждет более мощных языков, а вовсе не ограниченных подмножеств. Второе: есть осознание того, что такие попытки едва ли могут привести к успеху — ведь это была бы попытка укрепить структуру дома, построенного на песке. Есть вещи, которые надо продумывать до, а не после.

К.П.: Тогда что Вы думаете о Java? Этот язык очень быстро набирает популярность, но во многих отношениях, он как раз не слишком отличается от того "очищенного" C++, о котором я говорил.

Н.В.: Серьезный ответ на этот вопрос потребовал бы слишком много времени. А если кратко, то: приобретенная Java слава есть результат массированной рекламной кампании, а не его выдающихся технических достоинств.

К.П.: Не кажется ли Вам, что программистское сообщество слишком фокусируется на чисто технических вопросах, но забывает, что разработка ПО — это прежде всего человеческая деятельность? К примеру, я полагаю, что одна из причин, почему BASIC и Си стали столь популярными, коренится в том, что в них сравнительно немного ограничений против "локальных решений" (эвфемизм для "заплат"), которые вносятся на поздних стадиях цикла разработки. Мы все знаем, что перед тем, как приступать к кодированию, следует провести тщательное проектирование. Но мы также очень хорошо знаем, что в большинстве случаев менеджмент не склонен сразу платить за достоинства ПО, проявляющиеся в долгосрочной перспективе. И это одна из причин того, что далеко не все с энтузиазмом адаптируют ООП. Если же система не спроектирована качественно, то операции по внесению "заплат" на позднейших стадиях выполняются столь часто, что становятся рутинными. Естественно, что языки, которые позволяют без больших проблем это проделывать, становятся более широко используемыми, чем те, что требуют больших инвестиций в предварительное проектирование. Стоит ли нам поэтому так нападать на программистов с упреками в неряшливости и прочем, если вся их вина в том, что они на самом деле просто приняли правила игры, действующие в реальном, а не в идеальном мире? Не лучше ли вместо этого рассматривать языки программирования в контексте той деятельности, для которой они предназначены?

Н.В.: Конечно, разработка ПО — это техническая деятельность, проводимая людьми. Не секрет, что среди других вещей люди страдают от несовершенства, ограниченной надежности и нетерпения. Добавьте к этому запросы на более высокую оплату труда с одной стороны и требования более быстрого достижения результата — с другой. Однако работа, которая делается в спешке под постоянным давлением срока выброса продукта на рынок, неизбежно приводит к неудовлетворительным результатам — к продукту с дефектами. Существуют много более лучшие методы проектирования ПО, чем те, которые повсеместно используются, но им редко следуют. Я знаю одного очень крупного производителя ПО, который декларировал, что у него проектирование занимает 20% от времени разработки, а (условно говоря) отладка — остающиеся 80%. Те, кто ратует за обратное соотношение этих величин, не только убедительно доказали, что это реалистично, но и показали, что это улучшило бы тусклый имидж фирмы. Почему же, несмотря на это, подход с 20% времени на проектирование, по прежнему в фаворе? Да потому, что чем меньше времени вы проектируете, тем быстрее ваш продукт может оказаться рынке. И рыночные обзоры показывают, что по большой части потребитель рассматривает несовершенный, но стремительно возникший продукт как более привлекательный по сравнению с более стабильным и зрелым, но появившимся позже. Кого винить в таком положении дел? Программистов, превратившихся в хакеров? Менеджеров, находящихся под давлением не желающего ждать рынка? Бизнесменов, вынужденных быть рабом величины своей прибыли? Или потребителей, верящих в обещанные чудеса, которые будто бы достижимы с помощью нового продукта?

К.П.: Есть ощущение, что многие современные языки программирования перегружены функциональными возможностями, из которых обычно используется процентов 20, а то и меньше. Что показательно: большинство этих возможностей так или иначе увеличивают гибкость, а не безопасность. Например, старое правило "декларируй до использования" было признанием того, что программисты делают ошибки, и воплощение этого правила в жизнь позволяет компилятору их обнаруживать. Аналогично, конструкция "obsolete" в языке Eiffel является признанием того, что некоторые части кода устаревают; однако, без выдаваемых во время компиляции предупреждений, большинство программистов будет продолжать полагаться на эти части. У кого есть время на чтение комментариев и документации, когда продукт надо выпускать немедленно? Однако, в то время, как идут, к примеру, горячие дебаты относительно полезности конструкта "finally" в C++, я видел очень мало публично проявляемой заинтересованности в том, как защитить программистов (как людей) от них самих. Что Вы думаете по этому поводу?

Н.В.: Богатство функциональных возможностей во многих современных языках — это действительно проблема сама по себе, а не решение других проблем. Избыток возможностей — это еще одно следствие веры многих программистов в то, что ценность языка пропорциональна количеству этих возможностей (как я это называю — это вера в "колокольчики и свистки"). Однако, мы знаем, что будет лучше, если каждое базисное понятие представляется единственной, специально для этого предназначенной конструкцией. Это не только сокращает усилия по изучению языка, но и сокращает размер его описания, что, в свою очередь, помогает избежать несогласованности и неправильного понимания. Поддержание языка максимально простым и регулярным всегда было приоритетом в моей работе: описание Pascal занимало около 50 страниц, Modula — около 40, а Oberon — и вовсе 16. И я рассматриваю эту тенденцию как прогрессивную. Истинная ценность языков программирования зависит от качества и практичности их абстракций. Пример — абстракция, называемая "число" или абстракция "логическая величина", замещающая конкретную строку битов. Ценность такого рода абстракции основывается на ее целостности. В случае чисел должны быть применяемы только арифметические операции, независимо от того факта, что логические операции, в принципе, также могут быть применимы к битовым строкам, представляющим числа.

Н.В.: Другой случай — это понятие массива с заданным числом элементов, идентифицированным порядковыми числами (индексами). Должно быть гарантировано, что ни к какому элементу не может быть доступа с некорректным индексом, несмотря на факт, что вычисленный результирующий адрес будет указывать на некоторую существующую ячейку памяти — вероятно, хранящую другую переменную. Абстракция, о которой мы не устаем говорить, — это важнейшее понятие типа данных, и мы указываем, что его ценность основана на том, что компилятор будет проверять, соблюдаются ли правила, управляющие типами, и именно компилятор будет гарантировать целостность абстракции. Если система не в состоянии это обеспечить, если она допускает выполнение логической операции над числами или не способна идентифицировать доступ к массиву с некорректным индексом — просто привожу два примера для иллюстрации — то она вряд ли может претендовать на титул "система с языком высокого уровня". Однако, наиболее широко используемые системы программирования как раз имеют эту самую природу, и предложенная абстракция может вполне быть нарушена — правда взамен добавляются бесчисленные инструментальные средства сомнительного достоинства, которые помогают понять, как эта абстракция на самом деле представляется внутри, с целью обнаружить, что же неправильно.

Н.В.: Особая ирония в том, что языки с фиксированными структурными моделями для операторов и типов данных, с конструктами для модуляризации и скрытия информации, многими рассматриваются как ограничивающие творческое начало (и даже мешающие ему!). Действительно, программирование на структурированном сильно типизированном языке требует намного более тщательного осмысления задачи, а это само по себе рассматривается некоторыми как большой недостаток (вспомним, что программирование рекламируется как легкая и необременительная деятельность). А ирония в том, что время, которое действительно можно сэкономить, если не следовать структурным правилам, будет затем потеряно в многократном размере — хотя, возможно, не самим разработчиком, а потребителем. Мой прогноз не слишком утешителен: не думаю, что можно ожидать здесь больших изменений. Понятия и языки, которые способны преобразовать коммерческое программирование в действительно серьезную инженерную профессию, существуют уже достаточное количество лет, но их мало используют. Такое впечатление, что разработка ПО будет еще некоторое время оставаться Эльдорадо для хакеров.

Н.В.: Я вспоминаю дискуссию на академическом семинаре где-то в середине 70-х, когда термин "software crisis" был предметом горячих дебатов, а концепция доказательства корректности программ воспринималась как возможное средство излечения индустрии. Профессор Тони Хоар (C.A.R. [Tony] Hoare), докладчик на этом семинаре, красноречиво представлял принципы и преимущества доказательств корректности — прежде всего, как механизма, могущего заменить процесс отладки. После длительной дискуссии, Джим Моррис (Jim Morris — глава кафедры информатики в Carnegie Mellon University и разработчик таких основополагающих принципов, как межмодульная защита и "ленивые" вычисления) встал и с обезоруживающей интонацией спросил: "Но Тони, что ты скажешь, если мы честно признаемся в том, что обожаем процесс отладки? Ты хочешь, чтобы мы отказались от такого удовольствия?".

Н.В.: Да, я убежден, что есть нужда в высококачественном ПО. И придет время, когда будет признано, что стоит вкладывать усилия в его разработку и в использование точного и структурированного подхода на основе безопасных, структурированных языков. Пока же, однако, я вижу лишь небольшую нишу для такого рода практики. Она будет расширяться вместе с увеличением количества тех, кто будет все громче выражать свое неудовлетворение некачественным программным продуктом.