Две культуры программирования

В сентябре-октябре 2005 г. произошли два важных события, которые можно назвать поворотным пунктом в истории европейского программирования.

11 сентября прямым рейсом из Цюриха в аэропорт Пулково прибыла небольшая швейцарская делегация в составе профессоров ETH Zurich Никлауса Вирта и Юрга Гуткнехта. Так начиналось Большое турне Вирта по России. За три с небольшим недели он посетил ведущие университетские центры страны: С.-Петербург, Москву, Нижний Новгород, Екатеринбург, Новосибирск, Томск.

В 71 год пойти на столь изматывающую поездку в далёкую Россию — это надо иметь большое мужество. И понимать, во имя чего всё это делается. Европейская наука проиграла в жестокой схватке с американским бизнесом. И теперь потребуется немало усилий, чтобы осознать всю глубину падения и постараться выправить положение. Без России и стран бывшего Союза это сделать практически невозможно.

Сразу после возвращения Вирта из поездки по России и Польше, 20-21 октября 2005 г. в Цюрихе прошёл Европейский саммит по компьютерной науке (European Computer Science Summit), главным организатором которого стал преемник Н.Вирта на посту декана факультета информатики ETH Zurich, ученик А.П.Ершова, автор языка Eiffel, французский учёный Бертран Мейер.

Впервые за последние десятилетия Европа заявила о том, что не намерена слепо идти по пути, который исповедует Америка, что у Европы есть свои богатейшие традиции в этой области и своё видение развития компьютерной науки. Алан Тьюринг (Великобритания), Конрад Цузе (Германия), Эдгар Дейкстра (Нидерланды) — все они закладывали основы основ современной ИТ-индустрии. Но их идеи во многом преданы забвению. Европа не вступает в конфронтацию с США в сфере программирования. Она хочет равноправного и взаимовыгодного партнерства, возможности влиять на формирование своей интеллектуальной элиты. Центром "мирной революции" стал ETH Zurich, один из старейших и самых престижных университетов Старого Света, в стенах которого учились и работали такие выдающиеся ученые и инженеры, как Альберт Эйнштейн, Джон фон Нейман, Конрад Цузе, Никлаус Вирт.

Прямым итогом этих двух событий стало формирование в России научно-образовательного некоммерческого проекта Европейского центра программирования (EuroProg, European Programming Centre), который ставит своей целью создание влиятельного информационного и координирующего центра в сфере компьютерного программирования для стран европейского региона, прежде всего, Восточной Европы, включая Россию, Беларусь и Украину. Он создаётся на базе событийного сайта Oberon2005, подробно освещавшего Большое турне Вирта по России. Формирование ведётся при участии Высшей Политехнической школы ETH Zurich, Института компьютерных систем при ETH (Цюрих, Швейцария), а также Института систем информатики им. А.П.Ершова (Новосибирск, Россия).

Фактически строится новая модель сотрудничества университетских и исследовательских центров Европы с привлечением независимых профессиональных программистов и представителей программной индустрии. Строится модель, компенсирующая острую нехватку деятельности научно-исследовательских институтов и опытно-конструкторских бюро в сфере программирования. Как и при создании А.П.Ершовым знаменитого новосибирского центра на базе Академгородка, включавшего триаду “образование-наука-индустрия” (НГУ, ВЦ СО АН СССР, НФ ИТМиВТ), большая роль уделяется подбору и концентрации важнейших информационных материалов в рамках общедоступной открытой некоммерческой библиотеки (Open Digital Library). Другими составными элементами проекта станут Музей программирования, Открытая энциклопедия программирования, инструментальный репозитарий, а также научно-производственный центр, координирующий ведение проектов и НИОКР в рамках

модели открытого исследовательского программирования (Open Research Programming), которая является альтернативой Open Source. Открытие Европейского центра программирования намечено на осень 2007 г.

Другим отголоском событий 2005 г. стало образование влиятельной некоммерческой и неправительственной организации Informatics Europe(первоначально под именем euroTICS), которая ставит своей целью объединить усилия образования, науки и индустрии Европы на основе европейских традиций информатики. Это традиционная форма ассоциации с центром в ETH Zurich, в которую уже вошли преимущественно университеты Западной и Восточной Европы, включая такие страны, как Швейцария, Франция, Германия, Австрия, Великобритания, Италия, Испания, Нидерланды, Дания, Россия, Украина, Болгария, Чехия, Румыния, Литва, Турция.

Европа существенно слабее Америки в отношении поддержки собственной индустрией академической науки и образования. Да, есть ряд крупнейших европейских компаний, которые так или иначе способны влиять на эти процессы: Airbus, Nokia, Ericsson, Philips, Thomson, Siemens, SAP AG и др. Но это влияние пока достаточно слабое. В то же время, нельзя не заметить, что центробежные процессы постепенно сменяются на центростремительные. В условиях феодальной раздробленности компьютерной Европы начался процесс собирания земель. Как известно, на Руси (в отличие от государств Западной Европы) централизация стала не следствием достижения зрелости буржуазных связей, а потребностью борьбы с внешним врагом. В сфере программирования ситуация очень напоминает историю борьбы с феодальной раздробленностью Руси. В условиях тотальной коммерциализации и однополярности мира не приходится ожидать кардинального изменения ситуации за счёт вызревания политико-экономических связей. Холодная война Америки против Европы привела к крушению основ европейской школы программирования, к тотальной коммерциализации образования и науки. Удары наносились, прежде всего, по инфраструктурным вещам:

1) языкам программирования;
2) железу;
3) кадрам (образование);
4) операционным системам.

Проф. Эдгар Дейкстра (Edsger Dijkstra, Нидерланды, 1930–2002), один из отцов-основателей программирования, так охарактеризовал ключевые события, которые привели к формированию однополярного мира и разрушению европейской культуры программирования: "Принятие в Германии Алгола-68 оказало парализующий эффект на немцев, подобный тому, которому подвергся Советский Союз, когда русские в конце 1960-х приняли решение разрабатывать свою новую национальную серию компьютеров на основе поразрядно-совместимой копии IBM 360. То была величайшая победа Америки в холодной войне".

Вкратце история противостояния Америки и Европы, на мой взгляд, выглядит так:

Алгол-60 — отправная точка размежевания школ программирования: американской и европейской (после совместных работ под эгидой IFIP Европа взяла на вооружение Алгол-60, Америка — Фортран);

клонирование IBM 360/370 в серии ЕС ЭВМ в СССР и странах СЭВ явилось гарантией технологического отставания советской супердержавы, сдерживавшей амбиции Америки;

затяжное принятие раздутого Алгол-68 – разящий удар по европейским языкам программирования;

вымывание европейских производителей компьютеров (ICL — Великобритания, Nixdorf, Siemens — Германия, Bull — Франция; ИТМиВТ — СССР).

падение роли IFIP, доминирующая роль американских ассоциаций ACM и IEEE CS;

гегемония американских стандартов высшего образования (ACM/IEEE Computing Curricula);

американизация культуры языков программирования (Бейсик, Кобол, Фортран, Си, C++, Java, C#);

американизация исследовательских и промышленных операционных систем (UNIX, DOS, Windows).

Примерно в течение 15 лет (1960–1975) центральной организацией, регулирующей процессы развития науки и индустрии в сфере компьютерного дела в мире являлась IFIP (International Federation for Information Processing, Международная федерация по обработке информации; штаб-квартира во Франции, теперь в Австрии). Она была организована на первом Всемирном компьютерном конгрессе (World Computer Congress) в Париже в 1959 г. под эгидой UNESCO.

Но IFIP была центром де-юре, а де-факто таковым стала американская ассоциация ACM (Association for Computing Machinery, 1947) вместе с примкнувшей к ней IEEE Computer Society (1946). Единственным противовесом им (в локальном масштабе) стало Британское компьютерное общество (BCS, British Computer Society, 1957).

Америка твёрдо шла по пути захвата рынков, устранения конкурентов и навязывания собственных взглядов на прошлое, настоящее и будущее компьютерных наук и компьютерной индустрии. Одним из ключевых моментов стало постепенное разрушение европейской школы программирования за счёт американской культурной экспансии, ставящей во главу угла коммерциализацию.

Отличительные черты американской школы программирования: тотальная коммерциализация; инновации с целью максимального извлечения прибыли; неизбежное снижение наукоёмкости ПО; формирование искусственной сложности ПО; экстенсивный рост (мощности оборудования, объемы систем, масштабы проектов).

Отличительные черты европейской школы программирования: научные исследования — основа основ; инженерные решения требуют продуманности и отказа от излишнего усложнения; контролируемый рост сложности систем; интенсивный рост (за счёт высокой культуры образования и разработки специального инструментария).

Одним из немногих, кто вовремя понял всю суть процесса деградации науки и культуры под звёздно-полосатым флагом глобализации применительно к компьютерной сфере, был голландский проф. математики, один из первых профессиональных программистов, апологет европейской школы Эдгар Дейкстра. Предметом исследований Дейкстры было именно программирование как интеллектуальный вызов, как процесс творчества и созидания, а не просто набор отдельных языков, приёмов или методов. В работе "Мои надежды на вычислительную науку" (1979, "My Hopes of Computing Science", EWD709) он писал: "Ассоциация ACM имеет специальную группу по языкам программирования (Special Interest Group on Programming Languages), но не по программированию как таковому; её новейшее периодическое издание посвящено языкам программирования и системам (Programming Languages and Systems), но не программированию. Программистское сообщество почти болезненно зациклено на программной нотации; оно делится по сути на такое же количество субкультур, каково число распространённых языков программирования; субкультур, ни одна из которых чётко не отделяет истинные проблемы от проблем, порождаемых исключительно соответствующим языком программирования (подчас превращаясь в религию)".

Искусственно раздутые религиозные войны в сфере языков привели к размежеванию программистской общественности. В этой обстановке властителями дум стали именно те языки, в которых было важно не технологическое совершенство, а доминантное рыночное положение, определяемое маркетинговыми войнами софтверных гигантов. По сути это

внешняя открытость при реальном размежевании. Языки всё больше стали превращаться из субкультуры в культурный слой и стали диктовать свою культуру.

Применительно к программированию микрокультуры разных индивидуумов сосуществуют и взаимодействуют друг с другом, образуя культурные слои. Человек стремится создать вокруг себя условия жизненного комфорта, а значит, интуитивно ищет родную среду. Эта среда должна иметь определённые характерные черты, привлекательные для родственного сосуществования микрокультур. Что в программировании выполняет роль платформы для слияния микрокультур? Традиции научных школ, парадигмы, языки, инструментарий, операционные системы? На мой взгляд, сначала произошёл отход от научных школ в сторону конкретных языков и стирания их исторических корней. Затем уход от компактных языков-ядер (Лисп, Пролог, Форт, Си, Паскаль) в сторону языков-оболочек, растворяющихся в инструментарии за счёт объёма и экстенсивного наращивания встроенных возможностей. Инструментарий в отличие от языка не имеет, как правило, чёткой спецификации, а значит, в программировании "судебная практика" стала главенствовать над "законодательством". Ныне фундамент культуры выродился в инструментарий. Если проводить аналогию с музыкой, то основой единения микрокультур в программировании служит даже не конкретное направление, а конкретная группа, конкретный исполнитель. Что для программирования весьма печально.

В работе "Два взгляда на программирование" (1975, "Two views of programming", EWD540) Дейкстра писал: "Я придерживаюсь того мнения, что программирование – один из наиболее сложных разделов прикладной математики, поскольку оно также является одним из наиболее сложных направлений инженерии, и наоборот. Когда я попытался разъяснить одному из моих коллег-математиков, почему я придерживаюсь этого мнения, он довольно бесцеремонно отказался выслушать мои доводы и вместо этого обвинил меня и моих единомышленников-компьютерщиков в том, что мы до сих пор не создали язык программирования, который сделал бы программирование настолько простым, насколько ему и подобает быть! Возможно, мне стоило бы спросить его, почему математики до сих пор не разработали нотацию, которая позволила бы любому, невзирая на отсутствие профессиональной подготовки, заниматься математикой?"

Программирование — сложное культурное, социальное и экономическое явление, а не просто синтез науки и инженерного дела. Это серьёзный интеллектуальный вызов, а не чисто механический процесс кодинга и "кройки по лекалам", как его пытаются преподнести сейчас. Дейкстра писал (EWD540): "В основном это не вина производителей компьютеров, которые желают вести дела так, будто они продают простейшую продукцию; и не вина руководителей программных проектов, которые предпочитают рассматривать деятельность программистов как простой и предсказуемый процесс; и не вина учебных заведений, которые хотели бы подготовить студентов к достижению гарантированного успеха. Это — следствие комфортной иллюзии, что Человек — лишь сложный автомат, иллюзии, которая, подобно наркотику, приносит своим жертвам кажущееся освобождение от бремени ответственности. Признание программирования серьёзным вызовом интеллекту вернуло бы полный вес этой ноши обратно на их плечи".

При этом по силе своего воздействия на мировую экономику программирование становится с каждым годом всё более значимым катализатором развития общества.
Мне нередко доводилось слышать, что программирование не имеет национальных и региональных черт. Это глубочайшее заблуждение. В программировании существуют разные школы, имеющие разные традиции, исповедующие разные ценности.

В центре синтеза "интеллектуального топлива" для мировой экономики стоит человек. Со своим опытом, амбициями, принципами, взглядами, предпочтениями. Если смотреть на программирование как на феномен культуры, то неизбежно приходишь к сопоставлению позиций Освальда Шпенглера и Михаила Михайловича Бахтина. Шпенглер ратовал за замкнутость культур. Мне же в отношении программирования

ближе подход М.М.Бахтина: существует единая культура человечества, состоящая из многих меньших, принципиально открытых культур. Шпенглер писал: "Как бы не убедительно воздействовал художник своими звуками и цветами, наблюдатель слышит в них лишь себя самого, если он на это не способен, произведение искусства остается для него бессмысленным". Сравните с Бахтиным: "Чужая культура только в глазах другой культуры раскрывает себя полнее и глубже (но не во всей полноте, потому что придут и другие культуры, которые увидят и поймут ещё больше). Один смысл раскрывает свои глубины, встретившись и соприкоснувшись с другим, чужим смыслом: между ними начинается как бы диалог, который преодолевает замкнутость и односторонность этих смыслов, этих культур". Иными словами, чем больше коммерциализация уничтожает ростки другой культуры, тем сильнее она содействует полному замыканию доминирующей культуры, которая лишается возможности полноценно понять и оценить самое себя. А это — неизбежный путь к вырождению.

О том, к чему ведёт американизация культуры программирования, Э.Дейкстра весьма рельефно вскрыл в своей работе "Почему американская компьютерная наука кажется неизлечимой" (1995, "Why American Computing Science seems incurable", EWD1209). Он пишет: "По жестокой шутке истории, впрочем, американское общество выбрало именно двадцатое столетие для того, чтобы становиться всё более и более нематематическим (кстати, явление, рассмотренное Моррисом Клайном и вызвавшее у него глубочайшее сожаление). Мы достигли парадоксального состояния, когда из всех так называемых "развитых наций" именно США сильнее всех зависят от программируемых компьютеров и хуже всех интеллектуально оснащены в данном направлении. Предположение о том, что проблема программирования может быть вылечена математическими средствами, мгновенно отвергается как совершенно нереалистичная. В результате разработка программ ограждена от возможности стать поддисциплиной информатики. Имеет место значительная озабоченность корректностью, но она почти полностью направлена на верификацию программ a posteriori, потому что опять же это легче укладывается в мечту о полной автоматизации. Но, разумеется, многие рассматривают верификацию a posteriori как установку телеги впереди лошади, потому что процедура "сначала программирование, потом проверка" поднимает насущный вопрос, откуда берётся программа, подвергающаяся проверке. Если же она выведена, то верификация сводится к простой проверке вывода. А между тем методология программирования, переименованная в "программную инженерию", стала настоящим раем для гуру и знахарей. Будучи лишённой того, что обычно рассматривается как основное направление компьютерной науки, американская компьютерная наука стала большим неудачником. И мы не вправе винить за это университеты, поскольку, когда промышленность, наиболее нуждающаяся в их научной помощи, неспособна понять, что это высокотехнологичный бизнес, лучшие университеты оказываются бессильны. Университетам следует быть более просвещёнными, чем их окружение, и они способны на это, но не очень стараются это показывать. В нынешней политической ситуации непохоже, что можно ожидать быстрых улучшений; в ближайшем будущем нам придётся жить среди предрассудков, подобных тому, что программирование – это "настолько просто, что им могут заниматься даже члены республиканской партии". А что вообще дала информатика как наука?

Небольшой экскурс в исторические достижения информатики как науки и своё видение первичной роли науки Эдгар Дейкстра дал в работе "Научная фантастика и научная реальность в информатике" (1986, "Science fiction and science reality in computing", EWD952). Он пишет: "Что же информатика делала в этой блистательной изоляции? Точнее говоря, что она делала, не теряя претензий на прикладную значимость? Да, сделала она немало; фактически, куда больше, чем я могу разъяснить в рамках данной лекции, но я могу хотя бы передать вам общие моменты. В 1960-е годы она разработала теорию синтаксического разбора, необходимую для поднятия уровня компиляторов выше

уровня поделок, напичканных ошибками, и превратив её в предмет, пригодный для обучения. Это было главное достижение: я, например, помню весьма отчётливо, как в 1962 г. те из нас, кто действительно написал компилятор, выглядели в глазах остальных как некие полубоги. Этого никогда бы не произошло, если бы мы со временем не научились давать формальное определение синтаксиса компилируемого языка: без подобного формального определения слишком сложно было бы определить существование проблемы компиляции. Теория конечных автоматов и теория сложности были разработаны, чтобы задать основные количественные границы того, что в принципе может быть вычислено; опять же, в основе этих теорий лежит весьма формальный постулат о природе вычислений, постулат, без которого эти теории не могут существовать. Для разработки операционных систем была поставлена и решена проблема синхронизации процессов; были доказаны первые теоремы об отсутствии тупиков; первой предпосылкой этого достижения стало формальное определение явления, интуитивно известного как тупик (deadlock). В 1970-е годы центр внимания сместился от синтаксиса к семантике, сначала к детерминированным последовательным программам, а вскоре охватил также сферу недетерминированности и параллельности. Я не буду подробно описывать различные направления: они простираются от модели типизированного лямбда-исчисления до разработки преобразований программ с сохранением семантики. В этом десятилетии программы стали самостоятельными математическими объектами.

Кратчайший способ уловить изменение направления внимания – это, пожалуй, отметить, что если раньше задачей программ было управлять поведением машины, то теперь ею стало выполнение наших программ. Верификация и разработка программ развились в разделы формальной математики до такой степени, что теперь уже не считается безответственным опубликовать программу, не испробовав её на компьютере. Что ж, это далеко не полный обзор того, как информатика стала наукой, и я приношу свои извинения всем неупомянутым мной, кто внёс свой вклад в её становление. Надеюсь, что он всё же достаточно полон, чтобы донести до вас аромат квинтэссенции самой дисциплины. Она стала замечательной дисциплиной, поскольку разделение между "чистой" и "прикладной", столь традиционное для многих других дисциплин, совершенно поблекло и существенно утратило своё значение. Роскошь работы в окружении, в котором различие между чистой и прикладной наукой лишено смысла, — это, пожалуй, ещё одно признание того факта, что компьютер общего назначения действительно заслуживает эпитета "общего назначения". Она обладает всей пикантностью чистой математики, будучи более формальной, чем многие другие отрасли математики. Она не может избежать такой формальности, поскольку любой язык программирования, будучи интерпретируемым механически, представляет своего рода формальную систему.

В то же время она обладает всей прелестью прикладной математики, поскольку огромная мощность современных компьютеров даёт такие возможности для создания хаоса, что её методы необходимы, если мы не намерены попасться в ловушку сложности, которую сами же и создали. Научиться не попадаться в собственноручно произведённую ловушку сложности, сохранять вещи достаточно простыми и научиться достаточно эффективно мыслить о своих разработках — вот центральная задача информатики. Это также было осознано больше десятка лет назад, когда "разделение задач" стало находкой программной терминологии. Заметьте, пожалуйста, что путь, которым информатика пробивала свою нишу, был сам по себе примером успешного "разделения проблем".

Каждый раз, когда мы вкладываем уйму умственной энергии в тщательную разработку любой дискретной системы, мы делаем это не просто для удовольствия: мы всегда надеемся, что результат наших усилий будет использован во благо другим. Мы надеемся, что он будет удовлетворять нужды, соответствовать ожиданиям и доставлять удовольствие своим пользователям. В донаучный период разработки систем неформальное понятие "удовлетворения пользователя" было единственным приемлемым критерием качества программного обеспечения. Недостаток "удовлетворения

пользователя" как критерия качества состоит в том, что это не техническое понятие: он не задаёт технического направления разработчику и, кроме того, может быть достигнут другими средствами, помимо технических, например, агрессивной рекламой или промывкой мозгов. К науке это не имеет ни малейшего отношения. Задачи должны быть разделены, и тут на сцене появляются функциональные спецификации. Роль формальной функциональной спецификации – просто служить логическим барьером между двумя совершенно разными проблемами, известными как "проблема удовлетворённости" и "проблема корректности". Проблема удовлетворенности касается вопроса, соответствует ли система, отвечающая таким-то и таким-то формальным спецификациям, вашим ожиданиям и надеждам.

Проблема корректности касается вопроса, соответствует ли данная разработка таким-то и таким-то формальным функциональным спецификациям. Логический барьер был необходим, чтобы поставить проблему корректности перед информатикой: он изолирует уютную нишу информатики от проблемы удовлетворенности, решению которой наука мало чем способна помочь. Заметьте, пожалуйста, что я не утверждаю, что одна из проблем важнее другой; в конце концов, цепь не прочнее самого слабого звена. Однако я утверждаю, что проблема корректности представляет ту самую часть, которую нам удалось втиснуть в "тонкий пограничный слой" Бонди, где разумное применение научной мысли может принести пользу, тогда как неформализованная проблема удовлетворенности в действительности лежит на пределами компетенции науки. У вас могут быть самые разные проблемы, начиная с опасного перекрёстка и заканчивая такими, которые угрожают самому существованию целых поколений.

Однако наука никогда не решает ваши проблемы, она решает лишь свои собственные, и заключение о том, примете ли вы решение формальной научной проблемы как таковой в качестве приемлемого решения своей задачи, лежит целиком и полностью на вас. Другими словами, наука никогда не предлагает моделей реальности, она лишь строит свою теорию, и вопрос, примете ли вы своё восприятие действительности в качестве достаточно достоверной модели для этой теории, — полностью ваша проблема. Достоверность и реалистичность больше не являются научными понятиями, и учёный уступает право разглагольствования о них философам, пророкам и поэтам. С этой точки зрения роль науки довольно ограничена, на самом деле до такой неутешительной степени, что многие предпочитают закрывать глаза на ограниченность науки. Акцентировать внимание на этих ограничениях не принято в научной среде, поскольку помимо прочего это вызывает вопрос, а зачем тогда общество должно терпеть учёных. Это не шутка: все мы знаем, что если бы сегодня общество вдруг решило изгнать своих учёных, это было бы не в первый раз. И теперь, когда наука "вышла в народ", она также не столь популярна среди публики в целом.

Люди всегда питали противоречивые чувства по отношению к технологии, и с чем более мощными технологиями они сталкиваются, тем драматичнее становится двойственность этого отношения. Они чувствуют угрозу со стороны технологии сильнее, чем когда-либо прежде; в то же время их надежда на спасительную мощь науки и технологии всё больше крепнет. В старые добрые временя традиционного шаманства требовались только снадобья от всех недугов, Эликсир для вечной юности и Философский Камень для сотворения золота".

В программировании на смену учёным пришло шаманство и знахарство. Это характерная черта современного программирования. Вот почему очень важно понять, почему мы так плохо знаем достижения европейской школы, почему со скепсисом смотрим на науку и считаем себя настоящими профессионалами, которым и так всё ясно.