Современное ИТ: индустрия и образование. Обзор языка Ada.

С. И. Рыбин,
к. ф.-м. н., старший научный сотрудник НИИВЦ МГУ,
разработчик Ada Core Technoligies, Inc.

Настоящий материал представляет собой стенограмму лекции-экспромта, которая была прочитана С.И. Рыбиным перед слушателями спецкурса "Программное конструирование" на физико-математическом факультете Орловского государственного университета в октябре 2006 г., в период проведения в ОГУ международной конференции "Современные методы физико-математических наук", посвященной 75-летию факультета (см. http://conf75.phys-math.ru/, http://ermakov.metasystems.ru/photo.html). На лекции, кроме автора спецкурса И.Е. Ермакова также присутствовал преподватель кафедры информатики Е. Э. Темиргалеев.

Насколько я понимаю, все вы пришли сюда учиться тому, что называют словом программирование, чтобы потом в этой области работать. Вопрос: хорошо ли вы себе представляете, какова будет ваша работа? Представляете ли вы себе, что такое индустрия?

- Нет.
- Я так и думал.

Давайте я на эту тему попробую порассуждать, поговорить. Если говорить о программной индустрии, то вот что мне удалось понять... Я бы сказал такую неприятную, но единственно честную вещь: у нас, к сожалению, программной индустрии в том смысле, как её понимают в тех странах, где она есть, пока нет. Надеюсь, что когда-то возникнет. У нас есть (из того, что я видел) отдельные фрагменты этой индустрии, которые возникают в разных местах. Почему нет индустрии? Потому, что эти фрагменты ещё независимы друг от друга, к сожалению, нет той самой среды, в которой бы это всё "варилось" и воспроизводилось. Это в отличие от того, что мне доводилось видеть на Западе, и в чём даже удалось поучаствовать.

Однако всё равно люди работу находят - значит, в какой-то индустрии они работают. Я не знаю, какова ситуация в Орле, но то, что я видел в Москве... (Но Москва - это совершенно особенное место - про это говорят и изнутри Москвы, и снаружи ее...). Там народ более избалованный, в городе крутятся совершенно безумные деньги, при этом непонятно, что и откуда... Но, тем не менее, можно проследить совершенно чёткое разделение тех элементов индустрии, которые у нас остались: есть достаточно заметные, хорошо живущие на общем фоне фирмы, занимающиеся оффшорным программированием (аутсорсингом) - и есть всё остальное.

Что такое оффшорное программирование - это понятно? Другими словами, - это когда некоторый заказчик не хочет некоторую работу делать сам, он находит команду программистов: "Ребята, сделайте мне вот это..." - и "ребята" делают.

Как устроена работа в таких фирмах? Ситуация в этих оффшорных фирмах может существенно отличаться от ваших ожиданий. Чем? Такие фирмы - это на самом деле интеллектуальный конвейер. На входе - задания, которые поступают не важно от кого и не важно зачем. Задания эти фирме надо выполнить, чем больше она сделает, тем больше заработает денег; задача фирмы - зарабатывание денег и никакая другая. Фирма выполняет задания как на конвейере, как машины собирают - а здесь

программы делают... Так вот, к чему это я? Это я к тому, что если вы собираетесь идти в программирование и думаете, что будете заниматься творческой работой, то, вообще говоря, ничего подобного. Редко кому, редко когда это удаётся. К этому надо быть готовым заранее. Если вы думаете, что будете свободными людьми свободной профессии, то тоже ничего подобного. Во всех этих фирмах жесточайшим образом формализованы процессы разработки, то есть, расписано буквально по шагам выполнение заданий разработки. Всё начинается с разработки требований, которая проходит через определенные этапы, результаты доводятся до непосредственных разработчиков, изготавливающих определенные модули. Здесь тоже строго расписано - как надо документировать, как взаимодействовать с соседями и т.д. Вот такой конвейер у вас получится.

Проекты будут меняться достаточно часто. Недавно у нас на семинаре в МГУ выступала наша выпускница, которая работает в фирме IBS - это несколько тысяч программистов, занимающихся оффшорным программированием. Я специально поспрашивал, как у них жизнь устроена. Средний срок разработки проекта - полгода. Что такое полгода? На самом деле это - ничего: вы только успели войти во вкус - и у вас уже новый проект. При этом проект, в котором вы участвуете, не вы выбираете. Что вам дадут, то и будет.

Формализация разработки означает чёткое разделение ролей в команде. В больших организациях, как правило, есть системный аналитик. Знаете, чем он занимается? Вот представьте: приходит в компанию заказчик.. Он, на самом деле, еще не знает до конца, что ему надо. Задача системного аналитика - понять, что же ему на самом деле надо, перевести это на язык, понятный разработчику... даже не разработчику, а архитектору - и выдать некий документ, с которым уже может работать человек, который проектирует программную систему и заказчика никогда не видит. Этот архитектор представляет, какие у него возможности в команде, выдаёт общую архитектуру системы. Разделяет все требования на мелкие группы и отдаёт разработчикам, которые тем более не видят заказчика. Они по этим требованиям отрабатывают и сдают результат в отдел тестирования, которые берут напрограммированное, берут требования - и смотрят, получилось ли то самое, или не то самое. Вот что такое этот интеллектуальный конвейер.

Что из этого следует? То, что на самом деле кодировщик... Ну, кодировщик - это плохое слово... разработчик, человек, который создаёт код - лишь один из многих. Если взять список состава монстров типа IBS, то я думаю, и половины не наберётся. Есть аналитики, есть архитекторы, есть отдел обеспечения качества - эти люди занимаются тестированием и только тестированием.

Вопрос вам: попробуйте дать определение тестированию.

- Проверка на ошибки.
- Хорошо отвечаете. Цель тестирования какая?
- Выявить ошибки.
- Вот! Значит, вас правильно учат.

А очень многие даже у нас на факультете, когда задаёшь такой вопрос, отвечают, что надо проверить, что программа правильно работает. Ничего подобного. Тестирование - это принципиально разрушительный процесс: надо попытаться разрушить программу. И это тоже разновидность профессии. Откуда берутся тестировщики? Ведь вас учат программировать, учат ли вас тестировать? Нет...

И поэтому кадровый голод, который имеет место в индустрии, среди тестировщиков особенно острый. К сожалению, и у нас нет в университете систематических курсов по тестированию, мы пытаемся этот пробел закрыть специальными курсами, либо в рамках спецсеминара, чтобы мозги развернуть в эту сторону. Там на самом деле много своих хитростей.

Вот что такое та часть индустрии, которая существует в крупных городах - почему? Потому что заказчик какой хороший? Который из-за бугра. К сожалению, наши заказчики - даже такие корпорации, как Газпром и подобные - не хотят платить деньги. Иностранцы платят, а раз иностранцы, то такая разновидность бизнеса тяготеет к крупным городам. И на сегодняшний день, насколько я знаю, именно оффшорное программирование ближе всего к мировой индустрии. Есть, конечно же, и разработки свои собственные, есть успешные команды, но их гораздо меньше.

Если посмотреть на то, что происходит в индустрии вообще в мире и не ограничиваться только оффшорными фирмами (кстати, на Западе таких фирм тоже полно), рассмотреть, в том числе и команды, которые разрабатывают рыночные продукты либо с целью их продать, либо еще что-то... То что можно сказать про успешные команды? Успешные не в том смысле, что зарабатывают безумные деньги, а в том, что существуют достаточно долго, люди в них работают и получают не только деньги, но и удовольствие. Перво-наперво можно сказать совершенно точно, что время гениальных кустарей заканчивается, имеет место индустрия с технологиями, процессами и т.д. Практически везде жёстко всё расписано - шаг вправо, шаг влево - и ты уволен. Такая ситуация везде, то есть практически во всех успешных фирмах применяется систематический подход к процессу разработки с очень жёсткой дисциплиной.

Если взять нашу компанию AdaCore, которая уже более десяти лет занята разработкой и сопровождением системы программирования GNAT на базе языка Ada, у нас достаточно нетипичная компания. В каком смысле - если вы возьмёте любую книгу по тому, как организован процесс разработки, то увидите: там аналитик, архитектор, и т.д. У нас этого нет. Мы как-то ухитряемся десять лет жить, не имея отдельно выделенного звена менеджеров. Разрешается использовать какие угодно инструменты, работать в какой угодно среде, у нас продукты многоплатформенные. Но в то же время у нас существуют достаточно жёсткие внутрифирменные стандарты, как мы должны работать. Например, если возникает какая-либо проблема, то ты должен послать по определённой форме электронное письмо, чтобы эта проблема немедленно получила номер, попала в соответствующий список открытых проблем, проблему немедленно получает человек, за неё отвечающий, и т.д. Когда проблема закрывается, ты не можешь её просто так закрыть. Ты обязан написать тест, который проверял бы, что этой ошибки нестало, и не просто написать тест, а положить его в некую базу данных тестов, и с этого момента этот тест будет применяться ко всем следующим версиям технологии - при каждой сборке продукта, сборка происходит каждый день на всех платформах. Вот сколько всего вокруг казалось бы маленькой ошибочки. Эта ошибочка ещё должна быть аккуратно задокументирована, а после того, как была исправлена, еще и снабжена тестом.

Везде, в любой успешной индустрии вольницы нет. Есть жёсткая дисциплина. Эта дисциплина может быть идиотской на уровне того, что менеджер раздражает, эта дисциплина может быть, как у нас, не раздражающей, но нарушать её нельзя, иначе ситуация немедленно выходит из-под контроля. Практически везде используются

многочисленные внутренние стандарты, например, стандарт на стиль кодирования - казалось бы ерунда, как строчки разбивать, какие отступы делать... Ничего подобного. Если дойдёте до индустрии, то очень быстро увидите, что создание нового кода - это очень малая часть работы. В основном приходится работать с уже существующим кодом, и может быть, не вами написанным, так вот, вы не должны замечать, что этот код написан не вами, он должен выглядеть точно так же как ваш код. Мы с этим впервые столкнулись, когда у нас в университете был проект разработки компилятора C++ с некими голландцами, и они на нас спустили кучу всего, включая свои внутрифирменные стандарты на кодирование. Наши ребята умные, они сказали: "Что это за глупость такая"? Это неудобно, это некрасиво. Написали голландцам письмо. Они ответили: "Да, это некрасиво, но у нас уже этот стандарт десять лет используется, вы что, с ума сошли - его менять?" Не важно, какой стандарт используется, главное, чтобы он был - и у всех один и тот же. И это - абсолютно необходимое условие.

Что ещё сказать про индустрию такого, чего вы можете не ожидать? Как я уже говорил, создание своего кода - это очень малая часть работы. Вам крупно повезло, если вы сидите весь день и занимаетесь творчеством - свой код пишете. Как правило, основная работа в индустрии - это работа с чужим кодом, или со своим, но написанным полгода назад - а это всё равно, что чужой - уверяю вас...

Кто из вас умеет, а главное, любит документировать свой код? Если честно, то бюсь, что таких мало... В индустрии недокументированный код - хуже, чем отсутствие кода. Это - вредительство, чистой воды вредительство. Якобы задача решена, потом в него лезешь и не понимаешь, что там написано. Документированный код - это совсем другое дело.

Вот как началось моё вхождение в индустрию: есть язык Ada, вы про него немного знаете. Есть такая штука - semantic interface specification. Это некая библиотека, которая позволяет вытаскивать синтаксическую и семантическую информацию об Ada-программах. Я её реализовал. Как она реализуется? Естественно, она реализуется на внутренних структурах данных компилятора. Надо влезть в компилятор... Вот представьте себе... сложность языка представляете, компилятор написан и эксплуатируется уже много лет, и я в него влез - и успешно влез. И ASIS уже много лет является частью нашей индустриальной технологии, работает, им пользуется множество людей, на его основе сделано множество инструментов. Это не потому, что я такой умный, а потому, что этот код был составлен очень аккуратно, тщательно документирован, красиво структурирован - это очень важно, и поэтому оказалось возможным человеку со стороны прийти, напрячься и влезть внутрь. Очень важно понимать, что тот код ценен, с которым можно дальше работать другому человеку. У нас компания небольшая, если собрать со всего белого света - технических работников меньше шестидесяти человек, и каждый понимает, что именно так и надо, а в больших компаниях, где набирают программистов через кадровые агентства, если вдруг выясняется, что человек пишет недокументированный код, он сразу же перестаёт работать в этой компании.

Ещё раз хочу обратить ваше внимание на то, что в индустрии существует профессий больше, чем та, которой вас учат. Кроме аналитиков, архитекторов, тестировщиков существуют ещё профессии, которые можно назвать "не наши" - маркетинг и sales, т.е. отдел продаж. Что такое отдел продаж в нашей области? Вам надо найти того, кому ваш продукт интересен, понять его проблему и объяснить этому человеку, что ваш продукт, ваше решение для него полезно; с учётом того, что попытку повесить ему

лапшу на уши с целью продать сразу поймут. Какой квалификацией должен обладать такой человек? Должен ли он уметь разрабатывать? Не знаю... Но можно ли взять какого-нибудь экономиста? Скорее всего, нельзя. Откуда возьмутся такие специалисты? Такие люди тоже нужны. Имейте ввиду, что в индустрии может оказаться востребованным.

Ещё один интересный момент, лично для меня это был шок. Случилось мне долгое время дружить со Швейцарским политехническим университетом в Лозанне. У них два известных университета: в немецкой части это Цюрихский технологический, где работал Вирт, и он знаменит Паскалем и Обероном, а во французской части - Лозаннский и он знаменит именно "адскими" достижениями. Одно время я туда часто наезжал, и как-то нашёл нашего бывшего эмигранта, который там уже давно работает, и случился у нас с ним разговор за чашкой кофе про космос. И он как-то между делом небрежно сказал: "Был у вас там какой-то "Буран..." Мы там были вдвоем с Евгением Александровичем Зуевым - мы тут же взвелись: "Как это *был*? Наш "Буран", да он..." На что был вопрос: "Скажите, а он у вас летает?" Мы притихли.

- А летал?

Мы ещё больше притихли. Вроде два раза запустили, беспилотные испытания.

- Значит и не было у вас никакого "Бурана", и нету.

Вот этот принцип, летает - не летает, он совершенно железный. В индустрии присутствует ровно то, что используется и работает. Любые ваши идеи, любые ваши проекты, покуда они не летают, - это пустой звук.

Какое всё это имеет отношение к обучению? Как учиться таким профессиям, как готовить себя к работе в такой жёсткой и непонятной среде?

Еще такой момент... Идет совершенно сумасшедшая смена технологий. Даже если у вас оффшорная деятельность, то - вот у вас, допустим, идёт проект на C++, вы в нём уже натренировались, и вдруг новый проект, и вам говорят: "Забудь C++, Visual Basic у тебя". Запросто. И никто не будет думать о том, что рядом сидит команда, которая на Visual Basic'е уже работала, если она на середине проекта - то что, перебрасывать, что ли?

Вопрос - как учиться, чему учиться? Понятно, что вы уже в рамках какой-то учебной программы, чему вас учат, тому и будете учиться, но для себя вы тоже должны иметь некоторые ориентиры, понимать, что важно - что не важно, куда вкладываться - куда не вкладываться... Человек, который попадает в индустрию, начинает рано или поздно начинает об этом думать. Приходилось с коллегами обсуждать это и мы пришли к схожему пониманию того, как и чему стоит учиться. Сейчас я попытаюсь это сформулировать.

Начнём с фразы Вирта, который в прошлом году приезжал в Москву, выступал с лекцией в Политехническом музее, был у нас на факультете. Естественно, полная аудитория студентов, его пытают разными вопросам - и был такой вопрос: "Как вы считаете, какая основная проблема в нашей профессии?" Он отвечает: "Времени не хватает." Хронически всегда не хватает времени, чтобы подумать, чтобы выбрать правильное решение, как лучше реализовать проект - вот к этому тоже готовьтесь.

Всё стремительно меняется. Причем меняются не только инструменты, но и техника. Сейчас стремительно дешевеет Flash-память. Сейчас Flash-карту на 2Гб можно купить на каждом углу. Что будет дальше? А что это означает? Это означает, что через какое-

то время в области встроенных систем (которых у вас в кармане, как минимум, одна штука лежит, видимо) уровень сложности возрастёт на порядок. Какое там будет программное обеспечение, откуда оно возьмётся - а кто его знает...

Чему и как учиться? Бессмысленно в такой ситуации изучать детали конкретной технологии: вы закончите учиться, а там уже всё другое. Стоит учиться изучать детали. Вот чем хорош университет: он учит учиться. У нас недавно собирался наш выпуск, люди работают в самых разных отраслях, все в один голос сказали: "Спасибо университету, что научил учиться". Не свисточкам и звоночка, конкретным Турбо, Вижуал, не знаю там что... А надо научиться быстро их освоить, если Турбо-Вижуал поменялся на что-то другое...

Если не учиться свисточкам и звоночкам, то чему учиться? В нашей профессии, несмотря на то, что она стремительно меняется, все-таки можно выделить некие концепции, некие принципы, которые живут существенно дольше, чем конкретные версии конкретных систем. Например, задам вам такой вопрос: что такое тип данных, вы знаете. Попробуйте дать определение понятию "тип данных". Давать определения, даже если вы не преподаватель, даже если вы не пишете научную работу, это очень полезно. Это помогает узнать, понимаете ли вы, что это такое, или не понимаете.

Что такое тип данных? Вы же знаете, что это такое, попробуйте сказать это одним предложением. (молчание) - "Область занимаемой памяти?" Отлично, варианты ещё есть? (молчание) Вот смотрите - вы знаете, что такое тип данных. Единственная версия - область занимаемой памяти. Я вам дам вторую версию: множество значений при совокупности операций. Оно вам нравится как - больше, меньше? - "Ну, наверное, надо еще знать, как оно представляется в памяти..." А вот теперь я вам скажу такую вещь, что тип данных на самом деле есть описание содержательной роли объектов данных, которую они играют в программе. Как вы на это смотрите? Концепция строгой типизации, она вообще зачем нужна? Например, вы пишите некую программу - автопилот чего-то, что летает. У этого автопилота есть понятие высоты и понятие скорости. Они есть, если говорить про область памяти, если говорить про множество значений при совокупности операций - то это число с плавающей точкой. Правильно? А если говорить про содержательную роль, то это высота и скорость. Если вы описываете разные типы данных для высоты и скорости, зачем вы это делаете? А чтобы, упаси Бог, в первом часу ночи, когда вы в бреду горячечном, не успеваете - если вы случайно высоте скорость присвоите, что будет? Если у вас это область памяти - всем на это наплевать. Если у вас множество значений при совокупности операций - всем наплевать. Если у вас разные содержательные роли, то вас должен поймать компилятор. Именно для этого нужна концепция строгой типизации. Чтобы этой вашей ошибке не дать дойти до работающей программы, чтобы вас отругали раньше. И чем раньше вас обругают - тем легче вам будет жить...

Вот смотрите: есть детали - как это всё в памяти располагается, какие операции возможны, и т.д., а есть концепция: тип данных, как содержательная роль объекта в решаемой задаче, и эта концепция никуда не уйдёт, как бы языки не менялись. Какие-то языки следят за этой ролью более строго, какие-то - менее строго. Менее строго - это что значит: смотрим - присваивание высоте скорости... думаем... так, и то, и другое - вещественное число с плавающей точкой... а, ну плевать, преобразуем. Вот вам разница между языками.

Ещё такой вопрос: про объектно-ориентированное программирование знаете? Знаете.

Я получи кучу удовольствия несколько лет назад, принимая госэкзамен у нашего пятого курса. Что такое госэкзамен - ну, это вам еще предстоит. Грубо говоря, на пятом курсе вы сдаете экзамен вообще про все, чему вы эти годы учились. Естественно, там никто не помнит никаких деталей - причем не помнит не только сдающий, но и экзаменатор. Ни студент, ни экзаменатор не помнят всех деталей, потому что невозможно всё помнить. И как раз речь идёт о таком принципиальном моменте: у человека спрашивают: как устроено ООП в Паскале, в Дельфи? - рассказывает; как в C++? - рассказывает... А зачем нужно ООП? - всё. А как вы считаете?

А если я вам скажу, что ООП позволяет добавлять к системе новые свойства, не затрагивая тех клиентов, которые пользуются только старыми - это будет ответом? Вот этого ответа мне от наших пятикурсников добиться не удалось. Если посмотреть на это ООП, спрашиваю: "Что это такое?", мне начинают про классы рассказывать. Существуют языки, такие как Ada или Oberon - в них нет классов. При этом есть полноценное ООП. Когда про это говоришь - всё - человек останавливается.

Опять же, разница между свисточками и звоночками: то, что в Ada представляется теговым типами, то в C++ синтаксически оформлено в классы - это свисточки и звоночки. А вот то, что задача ООП - это дописывать новое, не разрушая старого - это концепция. Эта концепция присутствует во всех ОО-системах. Один раз её поняв, вы можете легко перемещаться между конкретными реализациями. А если вы зациклились, что ООП - это там, где написано слово class, то вот там вы и застряли.

Следующий момент. К тому, что Вирт сказал, что не хватает времени, я бы ещё добавил, что характеристикой нашей области является всё возрастающая сложность, причём возрастает она с колоссальной скоростью. Мы на конференции говорили на тему, что такое большая программа? Какую бы программу вы бы назвали большой? Попытайтесь оценить... Миллион строк?

Вот, например, наш компилятор - система программирования на базе языка Ada. Многоплатформенная, run-time отличается. Этих платформ с учётом кросс-платформ где-то две дюжины, плюс инструментарий: профилировщики, подсчёт метрик, pretty-printer, среда разработки. Порядка 40Mb исходного текста. 95% Ada, остальное - Си. Отнесли бы вы это к большой программе? Для современной западной индустрии это маленькая программа, причём очень маленькая. У нас есть клиенты, особенно авиационные, у них объём исполняемых файлов со всеми закрученными до упора ключами оптимизации доходит до 10 - 15 Mb. Просто задачи такие. Количество вложений шаблонов в реальных индустриальных текстах за десять запросто зашкаливает. Какими мозгами это можно понимать - я не знаю, но такие коды я лично видел.

Вывод: задача наша сложна, да ещё и какая плотность! Если вы пытаетесь сделать сложный самолёт, то это технически сложная задача. Плюс к этому у вас есть ограничения: металла надо много, огромное количетсво керосина для испытаний; ваш проект ограничен ресурсами. Чем ограничен программный проект, какими ресурсами? Да никакими, кроме человеческих: компьютер стоит копейки. Сложность, не ограниченная ресурсными ограничениями, имеет тенденцию расти бесконтрольно. Как я уже сказал, наш компилятор считается маленькой программой, а когда я был на первом курсе, у нас в качестве практикума было одно-единственное задание: реализовать сортировку каким-то из классических методов, и это считалось много. Теперь это делается у доски, в рамках половины занятия или семинара, у нас это был

целый семестр. То есть, темпы сумасшедшие, и сложность не собирается останавливаться. В реальных задачах сложность будет, от неё никуда не деться. Что с ней делать? Учиться ей управлять. Управляемый сложный процесс - это тяжело, но терпимо, неуправляемый - это провал. Пытайтесь то, что вам показывают, рассматривать в том числе как механизмы, инструменты борьбы со сложностью, её структуризации и управления ей.

Наверное, стоит задать ещё один вопрос: как вы считаете, программирование - это математика, или нет? - "Нет." Ну, вот я так скажу, что это - математика, и к ней вполне применимы математические методы, но... В свое время, ещё в Советском Союзе, была выдвинута концепция частично формализованных моделей. Слово "частично" здесь ключевое. Посмотрите на описание любого языка: синтаксис абсолютно формален, а описание языка программирования состоит из текста на естественном языке, и все попытки формального определения языка полностью - ни к чему не привели. Почему? Объект слишком сложный. Кстати, знаете ли вы, что в любом определении любого языка программирования содержится масса ошибок вплоть до серьёзных логических противоречий. В своё время, когда придумали язык Ada, безумное количество ресурсов в него вложили, - и люди со всего белого света находили ошибки в первой версии языка, был известен адрес, куда надо было посылать сообщения, соответствующие группы экспертов их анализировали. Был опубликован перечень дефектов стандарта Ada, он в три раза превосходил по объему стандарт самого языка. Первый в СССР ГОСТ на язык программирования был разработан, в том числе моими научными руководителями, максимально тупым, но в тоже время эффективным и полезным способом: переводом американского национального стандарта. Это был язык FORTRAN 66 года. Стандарт - книжка в 30 страниц. Переводчики знали язык досконально, потому что сами писали транслятор. После этого в течении нескольких лет успешно сдавались курсовые и дипломные работы на тему "Поиск дефектов в ГОСТе FORTRAN'а". Дефекты находились дюжинами, и никуда от этого не деться, просто объекты настолько сложны, что в описании всегда будут ошибки. Да, у вас математика, но какая математика? Возможности применения тех моделей, как в мат. анализе и формальной логике, - они ограничены. Это надо хорошо понимать. Принцип частичной формализации: покуда формализация работает - прекрасно, как только застряло - может быть, стоит остановиться, потому что, скорее всего, и не получится, а ресурсы вгрохаем...

Могу ещё поговорить на тему языка Ada. На конференции у нас был доклад на тему использования Ады в учебном процессе, поскольку как базовый язык для преподавания информационных технологий - язык очень хороший... Про Аду у нас в стране, к сожалению, либо вообще не знают, либо знают совершенно странные вещи. В своё время у нас была статья, где был перечень "адских" мифов.

Что такое Ada, откуда она взялась? Ada - язык, который возник в конце 70-х годов, то есть технология достаточно старая по своему происхождению. Возник язык потому, что Пентагон заметил, что происходит какой-то кошмар в области выполнения заказов по разработке программного обеспечения. Что значит кошмар? Систематический срыв сроков, превышение бюджета и падение надёжности продуктов. Был проведён анализ ситуации, это было как раз время "вавилонского столпотворения" языков программирования. Обнаружилось, что по контракту Пентагона используется четыре сотни языков высокого уровня. Сейчас невозможно это себе представить, потому что известные языки можно по пальцам пересчитать, но тогда это было так. Эксперты поняли, что ни один из этих языков никуда не годится. Что делать? Было принято

решение попробовать создать единый язык, который был лучше большинства существующих. И был запущен соответствующий проект. Несмотря на то, что все считают Аду военным языком - военные только запустили проект и профинансировали его. Выполняли его гражданские авторы. Материалы были открыты для печати. Язык разрабатывался интересным образом: начали с определения требований, в качестве ключевого требования была выделена надёжность создаваемого кода. Вторым по значимости требованием было осознание того, что программирование суть не создание нового кода, а сопровождение и модификация уже существующего, а потому программы должны в первую очередь быть понятны читателю. Сокращения, как в C, сходу отметались. Главное, опять-таки, чтобы ближе к полуночи, уставшему человеку, с замыленным взглядом было легко читать. Третьим требованием была общность, т.е. язык должен предоставлять адекватные средства решения для всех технологических проблем в области встроенных систем. Что такое встроенная система? Американцы определяли в своё время совершенно замечательно: встроенная система - это аппаратно-программный комплекс, предназначенный для решения задач, отличных от числовых расчетов. Очень важно, что встроенная система - это не то, что можно включить и выключить. Это система, которая живёт в пространстве и времени в рамках какой-то большей системы. Это может быть система управления кассовым аппаратом. А может быть - самолетным двигателем. Встроенные системы отличаются дополнительными сложностями: нестандартная среда, в том числе ввода-вывода, долгое время жизни системы. Самолет - лайнер - эксплуатируется лет 20, начинка его за это время не один раз изменится: то двигатель поменяется, то локатор, то еще что-то. А система-то боротовая одна, значит - то одно надо переписать, то другое... То есть, программное обеспечение часто меняется, меняется часто теми, кто в глаза не видел авторов...

В результате открытого конкурса, что интересно, победила французская команда. Т.е. Ада - это европейский язык. Ну, не французский, там была интернациональная команда, но основным автором был француз, Жан Ишбиа. Долгое время язык был официальным языком министерства обороны США. Ada был зарегистрирована как торговая марка, был составлен стандарт контроля трансляторов. И был введён закон: все военные заказы - только на Аде. В это дело было вложено много денег - от разных стран. Был "адский" бум в 80-е годы. Потом по разным причинам в Ada-индустрии был некий провал.

Можно на этом примере проанализировать, что такое наша индустрия: где наука, где технологии. Ada - торговая марка Пентагона, все военные заказы - только на Аде, лучше нет в нашем жестоком мире, чем военные заказы: гарантирована куча денег. Многие разработчики вкладываются в то, чтобы получить сертификат у Пентагона на свой компилятор Ады; всё это на мейнфреймах, никто не заботится о производительности, ресурсы огромные. И тут появляются персоналки - маленький ящичек, на котором можно играть в игрушки, набирать тексты, и т.д. Появляется простенький C, простенький Паскаль, реализуются на этих простеньких персоналках в совершенно рыночных условиях. Происходит существенный скачок в технологии компиляции: как в эти слабые персоналки впихнуть свои программы. Научились впихивать. Производители Ады заботились о сертификате Пентагона на мейнфреймах. Прозевали: выяснилось, что у каждого на столе стоит персоналка, в ней есть C, а Ады нет. Вот смотрите, как всё сошлось в одной точке. Ещё и техническая ошибка проектирования языка, которая здорово нагадила, заключалась в решении, что Ada - единый язык, и больше ничего не надо. И тут поняли, что сил не хватает: многие задачи уже решены на других языках, использовать бы их как-нибудь в своей программе... А

использовать нельзя. Поэтому многие стали отворачиваться от Ады. Когда пересматривали стандарт в 95-м году, спохватились - ввели такой интерфейс с другими языками... В частности, можно писать ОО-программу на Аде, в какой-то момент отобразить её в классы C++, продолжить развитие классов на C++, а потом вернуться обратно в Аду. Не понимаю, зачем, но язык такое позволяет. Но было поздно: упущен момент, когда народ от Ады отвернулся. Пентагон отказался от своих прав на торговую марку, был некий провал. В 98-м году язык был брошен в рыночную среду совершенно неподготовленным. Теперь постепенно индустрия Ады у нас растёт, но её не видно. Почему не видно? Потому что Ada - это не коробочный продукт; ниша, которую язык уверенно занимает - это большие встроенные системы. Например, в Париже есть ветка метро, где поезда ездят на автомате, без машиниста, полностью управляемые Ada-программой: заходишь в вагон, смотришь вперед - видишь тоннель, смотришь назад - видишь тоннель, кабины машиниста нет. И ездит - полностью управляемый Ada-программой. Авиационные системы - как вы увидите этот продукт, это не коробочный продукт. В то же время язык доказал свою универсальность - чем? Ну, вот у нас компилятор Ады написан на Аде, среда разработки тоже на ней написана, то есть Ada - вполне универсальный язык. Там, где работает C++, работает и Ada.

Стоит развеять некоторые мифы. Первое - что Ada - военный язык. Ничего подобного, Ada уже давно не имеет отношения к Пентагону. Второе - что Ada - мёртвый язык. Тоже ничего подобного: вполне используется, только не так виден, как Майкрософтовские коробочные продукты. Третий миф - что Ada - язык для специальных военных применений. Это не так, Ada - язык вполне универсальный. Четвёртый миф - что Ada - слишком сложный язык. Чтобы понять, что это не так, вы возьмите и положите рядом стандарты Ады и C++. Стандарт C++ заметно толще. Подсчитайте количество ключевых слов: в C++ их намного больше. В своё время, когда Ада возникла, был такой язык, ПЛ/1. Это был жуткий монстр. Тогда такая статья проскочила: и то и другое сложно, но если сложность Ады можно сравнить со сложностью часового механизма, то сложность ПЛ/1 похожа на прайс-лист огромного универмага. То есть в Аде управляемая сложность. Ещё один миф - что компиляторы медленные. Так было когда-то, теперь если взять какую-нибудь классическую задачу и написать её на Паскале, C, C++ и Аде, откомпилировать её индустриальных компиляторах и измерить производительность, то получится разница в десятые доли процента. В современной ситуации это - ничто. Причём неизвестно, кто выиграет - это от задачи будет зависеть. Ещё один миф - что Ada слишком дорогая. В какой-то мере это так. Бизнес-модели: у вас 1000 человек, которые заплатят по 10 рублей, или у вас 10 человек, которые заплатят по 1000 рублей. Ada относится ко второй ситуации. Да, дорогие компиляторы. У нас не компилятор продается, а техподдержка - она тоже недешево стоит. Но для университета существует возможность совершенно бесплатно получить полноценную версию нашей технологии с фирменной технической поддержкой. Существует возможность получить наш компилятор под лицензией GPL без ограничения функциональности. Это полноценный компилятор, из функциональности ничего не обрезано, кроме того, что позволяет, не нарушая закон, писать только GPL-программы. Бесплатно получить полноценную Аду каждый может совершенно законно.

Могу ещё поговорить на основе опыта полугодового спецкурса - о том, как язык может помогать писать надёжные программы и мешать ошибаться. Простой пример - оператор выбора *case.* Так вот, компилятор Ады не позволит вам откомпилировать программу, если не перечислены все возможные значения управляющей переменной и отсутсвует последний выбор "все, не перечисленные выше". Вы просто не сможете забыть ни один из вариантов. Вы свое намерение обязаны прописать явно абсолютно

везде. Вот такой мелкий пример.

Пожалуй, на этом я должен себя остановить. Я выдал вам много новой информации. Ну, половина - информация, половина - мои собственные ощущения о того, что происходит в нашей области. В свое оправдание могу сказать, что эти ощущения я проверял на коллегах, и, как правило, они совпадали. Так что, не я это придумал, точнее, не я один это придумал... Задавайте вопросы, пожалуйста.

Е.Э. Темиргалеев: *Как вы считаете, почему компания предоставляет университетам бесплатную поддержку компилятора?*

- Конечно, производитель заинтересован в подготовке кадров. Здесь задействована индустрия. Еще один момент. Если кто-то захочет попробовать Аду, то это не составит проблемы: если владеете Паскалем, то изучение ядра Ады - дело максимум одного дня.

Е.Э. Темиргалеев: *Еще одно уточнение... Можно сказать, что сейчас вышел новый стандарт Ada-2005. Если раньше были какие-то нюансы в C++ - если не говорить о надежности и читабельности кода - которых в Ada не было: например, множественное наследование, стандартная библиотека шаблонов и т.п., то сейчас эти моменты в новом стандарте учтены, и реализация Ada по новому стандарту предоставляет фактически все те возможности, которые предоставляют C++-реализации. В стандартную библиотеку включены контейнеры, сделано множественное наследование интерфейсов - и плюс повышена безопасность и т.п. В язык встроена поддержка распределенных вычислений... Т.е. те же самые возможности, что и в C++, плюс некоторые другие, которых тот не предоставляет. Так что есть смысл студентам над этим подумать. Кто особенно сильно увлекается C++ и не готов переходить на компактный Оберон, то, может быть, вам стоит заняться Адой и посмотреть на ее возможности. Ada и Оберон - языки одного Паскаль-семейства, и переход от одного к другому не представляет сложности.*