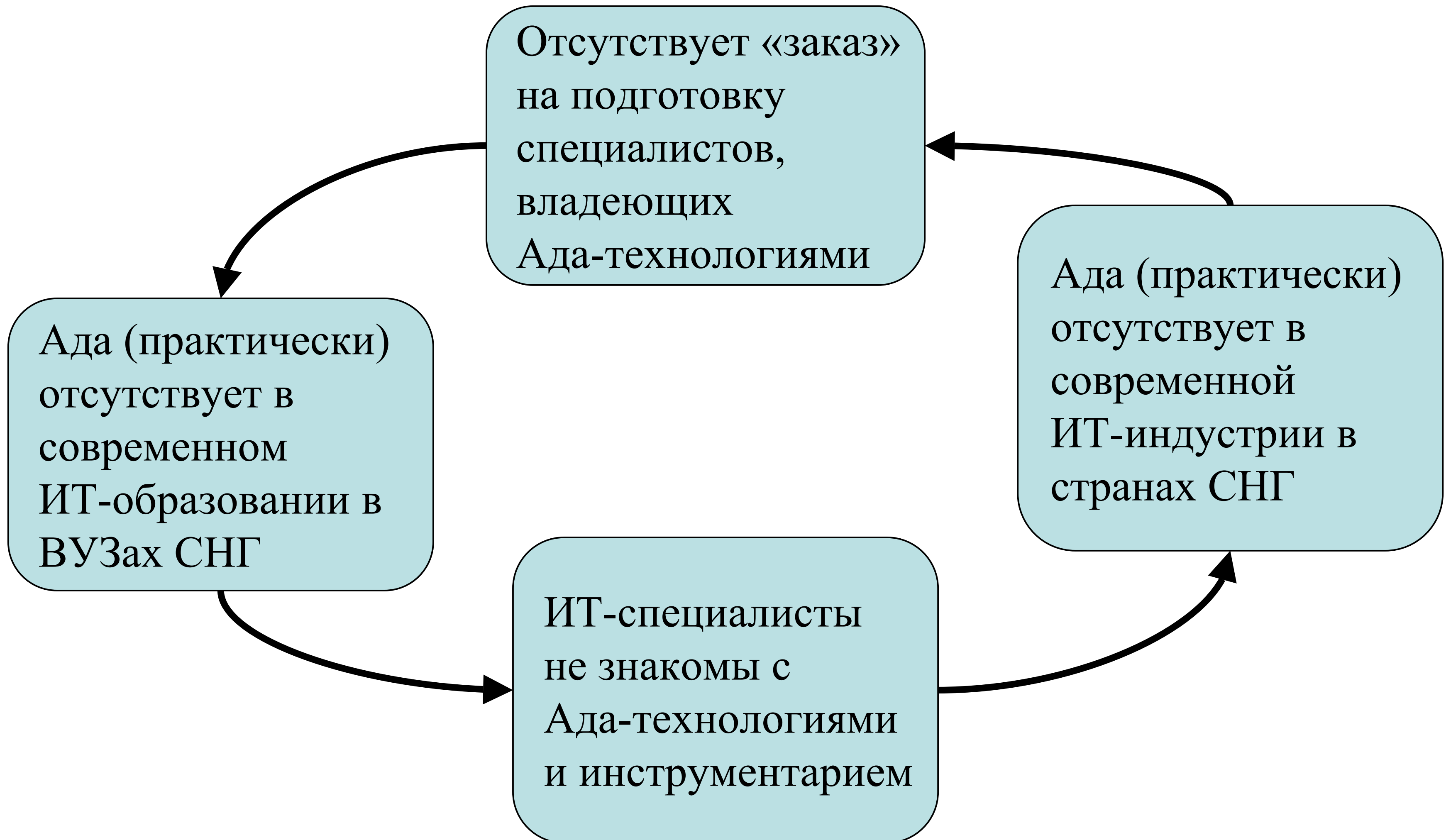
Ада –
перспективы использования в индустрии и образовании.

Сергей Рыбин
rybin@adacore.com

Вадим Годунко
godunko@adacore.com

Минск, 18 ноября 2009 г.

Еще одна попытка разорвать порочный круг…



И это в ситуации, когда Ада используется

- в таких областях, как:
  - организация воздушного движения
  - авионика (военная и гражданская)
  - средства связи и телекоммуникаций
  - энергетика
  - банковские системы
  - медицинские системы
  - военные (наземного, морского и авиакосмического базирования)
  - космические технологии
  - телевидение
  - транспорт
  - электроника
  - ...
- Такими компаниями, как:
  - Alenia
  - Alstom Transport
  - Ansaldo STS
  - BAE Systems
  - Boeing
  - EADS
  - European Space Agency
  - Eurocontrol
  - IPESOFT
  - JEOL
  - Lockheed Martin
  - MBDA
  - Philips Semiconductor
  - Raytheon
  - Rockwell Collins
  - SAAB
  - General Electric
  - Thales
  - Thales Alenia Space
  - ...

Почему Ада оказывается эффективным решением для индустрии?

- **Ада-технология – это:**
  1. язык программирования Ада, определяемый стандартом ISO/IEC 8652;
  2. реализация языка в той или иной системе программирования;
  3. техническая и информационная поддержка, которую может получить пользователь Ада-технологии;

- **Успех технологии – сумма (произведение?) успехов каждой из компонент;**
- **2-я и 3-я компоненты технологии практически не зависят от языка программирования;**

- **В чем преимущества Ады как языка разработки больших (встроенных) систем (реального времени) с повышенными требованиями к надежности?**

Ада – язык, ориентированный на создание надежного кода.

- **Современные индустриальные языки сопоставимы по предоставляемым возможностям, различаясь в том, как именно эти возможности предоставляются.**

- **Ада – единственный из современных индустриальных языков, созданных для обеспечения надежности больших программных систем – как при их разработке, так и при сопровождении.**

- **Ада предоставляет возможности, отсутствующие в языках-конкурентах (Си++, Джава):**
  - ***высокоуровневые языковые абстракции и соответствующие конструкции для программирования асинхронных процессов***

- **Производительность труда программиста на полном жизненном цикле программного продукта – один из важнейших принципов обеспечения надежности.**

“Hello, World!”

- **Язык был разработан в конце 80-х годов по заказу МО США;**

- **Стандарт ANSI (1983), стандарт ISO/IEC 8652 (1987/1995/2007);**

- **Ада – прямой потомок Паскаля:**

```
with Ada.Text_IO; use Ada.Text_IO;
procedure Hello_World is
begin
   Put_Line (“Hello, World!”);
   --  Это комментарий
end Hello_World;
```

Язык программирования и надежность - история.

- **Ада была создана как средство преодоления кризиса в разработке ПО по заказам Пентагона, который готов был разразиться в 70-е годы прошлого века, причина кризиса – в катастрофическом падении надежности как вновь создаваемых систем, так и модификаций существующих систем в ходе их сопровождения;**

- **Язык был разработан на основе систематической процедуры на основе перечня требований, основным из которых было требование по обеспечению надежности кода как в ходе его создания, так и в процессе сопровождения;**

Язык программирования и надежность - философия (1).

- **Создание программы (программной услуги) – не написание кода, а определение и использование моделей и абстракций, соответствующих объектам и процессам в решаемой задаче или проблемной области.**
- **Уровень и разнообразие базовых абстракций должен соответствовать уровню и разнообразию решаемых задач («принцип сундука»)**
  - готовые и удобные решения для часто встречающихся технологических потребностей;
  - удобные средства создания проблемно-ориентированных абстракций;
  - высокая производительность на полном жизненном цикле программной услуги;

Язык программирования и надежность - философия (2).

- «Все, что не разрешено – запрещено!» Язык реализует жесткую дисциплину **прогнозирования** (определения свойств абстракции при ее создании) и **контроля** * (использования абстракции в соответствии с определенными для нее свойствами);

- Чем раньше обнаружится ошибка (несоответствие использования абстракции определенным для нее свойствам), тем проще, быстрее и дешевле оказывается ее устранение;

----

* Кауфман В.Ш - Языки программирования. Концепции и принципы М.:Радио и связь, 1993

Язык программирования и надежность - философия (3).

- **Программист в основном занят сопровождением и модификацией чужого кода, а не созданием своего**
  - ясность и хорошая структурированность кода многократно важнее возможности быстро его написать!
- **Сложный инструмент редко бывает надежным**
  - предоставляя больше возможностей, чем Си++, Ада оказывается существенно более простым языком для изучения и понимания;
- **Все, что может быть определено – должно быть явно определено!**
  - умолчания и надежда на «здравый смысл» - один из основных источников недоразумений и ошибок!
  - определение Ады «замкнуто». По сравнению с Си/Си++, например:
    - правила видимости не требуют привлечения мифического понятия «пространства имен», а вполне обходятся синтаксическими конструкциями языка;
    - правила модульности и раздельной компиляции не требуют привлечения внеязыкового понятия «файл», а также обходятся синтаксическими конструкциями языка;

Язык программирования и надежность - философия (4).

- **Опережающая стандартизация**
  - Ада возникла и развивалась как ***стандарт*** языка программирования;
  - Средства контроля стандарта Ады были разработаны к моменту принятия первого стандарта языка – все индустриальные реализации Ады достаточно точно соответствуют той или иной версии стандарта, диалектов Ады в индустрии не было и нет;

- **Классификация ошибок в программе на уровне стандарта языка. Стандарт четко подразделяет все содержащиеся в нем требования на следующие группы:**
  - проверяемые во компиляции отельного модуля (нарушение требования означает, что компиляция не является успешной, в ее результате не будет создан объектный код);
  - проверяемые при сборке программы из успешно скомпилированных модулей (нарушение требования означает, что исполняемый файл создан не будет);
  - проверяемые при выполнении программы (нарушение требования приводит к возбуждению предопределенного исключения);
  - правила, нарушение которых реализация проверять не обязана.

  Средства контроля стандарта проверяют, что поведение реализации соответствует этой классификации!

Язык программирования и надежность – практика. О пользе ясного синтаксиса (1)

- **Опасность ошибиться и не заметить:**

Си - правильный код:

```
if (the_signal == clear)
{
  open_gates (...);
  start_train (...);
}
```

Но если перепутать “= “и “==“, получим формально корректный код:

```
if (the_signal = clear)
{
  open_gates (...);
  start_train (...);
}
```

В Аде такое невозможно:

```
if The_Signal = Clear then
  Open_Gates (...);
  Start_Train (...);
end if;
```

В Аде присваивание “:=“ никогда не может оказаться частью конструкции, являющейся условием и быть перепутано со сравнением “=“. В Аде операторы четко отделены от выражений.

Язык программирования и надежность – практика. О пользе ясного синтаксиса (2)

- **Опасность ошибиться и не заметить:**

Си - правильный код:

```
if (the_signal == clear)
{
  open_gates (...);
  start_train (...);
}
```

Если случайно поставить «лишнюю» точку с запятой, получим формально корректный код:

```
if (the_signal == clear) ;
{
  open_gates (...);
  start_train (...);
}
```

В Аде такое невозможно:

```
if The_Signal = Clear then ;
  Open_Gates (...);
  Start_Train (...);
end if;
```

В Аде не бывает неявных пустых операторов, которые «вдруг» появляются перед “;”. Данный код будет отвергнут как синтаксически некорректный

Язык программирования и надежность – практика. О пользе ясного синтаксиса (3)

- **Полезные «мелочи»**

```
procedure P1 is
   ...
   procedure P2 is
     ...
   begin
     ...
   end P2;
begin
   ...
   Bl1 : begin
      ...
      Bl2 : begin
         ...
      end Bl2;
      ...
   end Bl1;
end P1;
```

```
record
   I : Integer;
end record;
...
if Condition then
   for I in 1 .. 10 loop
      case Value is
         ...
      end case;
   end loop;
end if;
```

Язык программирования и надежность – практика. Прогнозирование и контроль.

- **Прогнозирование и контроль свойств элементов программы - основное средство обеспечения надежности:**
  - **каждая** используемая в программе сущность должна быть объявлена;
  - объявление **полностью** определяет свойства сущности;
  - **каждая** сущность может использоваться только в соответствии с ее свойствами, заданными в ее объявлении;
  - никакие **неявные** преобразования или изменения свойств сущности недопустимы;

- **Все, что не разрешено (объявлением сущности), запрещено (при использовании сущности);**

Язык программирования и надежность – практика. Строгая типизация в Аде (1).

- **Что такое тип данных в языке программирования?**
  - множество значений + набор применимых операций

  или все-таки
  - отражение содержательной роли объектов данных в проблемной области?
- **Типы данных и надежность программ - философия Ады:**
  - надежность программы возрастает, если программа максимально точно и подробно отражает модель проблемной области;
  - для каждого объекта данных важно определить его роль в (модели) проблемной области и проконтролировать, что каждый объект используется только в своей роли;
- **Концепция строгой типизации:**

  тип данных = содержательная роль объектов данных в модели проблемной области;

Язык программирования и надежность – практика. Строгая типизация в Аде (2)

- Все типы данных явно определены (объявлены) в программе, все они имеют имена и определения;
- Все объекты данных (и все их компоненты) имеют ровно один тип. Этот тип известен при объявлении объекта. Этот тип должен быть задан путем указания имени ранее объявленного типа данных;
- Все операции объявлены в тексте программы, все они имеют явно определенные типы параметров и результата (если он есть) Эти типы также должны быть заданы путем указания имен ранее объявленных типов данных.
- При вызове любой операции (присваивание тоже считается операцией!) проверяется, что типы операндов соответствуют заявленным типам параметров операции;
- Соответствие типов есть не соответствие структур значений, а совпадение имен (объявлений) типов;
- Все типы должны определяться статически (это никак не связано и никак не препятствует динамическому полиморфизму как части объектно-ориентированного программирования)
- **Неявные преобразования типов недопустимы**.

Язык программирования и надежность – практика. Строгая типизация в Аде – классический пример

```
type фрукты    is new Integer; --  тип с теми же свойствами, что
                               --  Integer, но ДРУГОЙ!!!
type Яблоки    is new фрукты;
type Апельсины is new фрукты;
...
Я1, Я2 : Яблоки;
А1, А2 : Апельсины;
Ф      : фрукты;
...
Я1 := 5;        --  можно
Я1 := Я1 + Я2;  --  можно
А1 := Я1 + А2;  --  нельзя - нет такого «+»!
Ф  := Я1;       --  нельзя - разные типы источника и получателя «:=»!

if  Я1 + Я2 > Ф then     -- и это недопустимо! Нет такого ">"
   …
end if;

-- А вот так – можно! Явное указание на то, что объект (временно)
-- меняет свою содержательную роль (явное преобразование типа):

Ф := фрукты (Я1) + фрукты (А1);
```

Язык программирования и надежность – практика. Строгая типизация в Аде:

(«Если нельзя, но очень хочется, то можно!» ((C)))

- **Иногда возникает необходимость преобразования типа между типами, вообще не имеющими друг к другу никакого отношения…**

- **Это можно сделать, но для этого придется специально настроить вот такой шаблон:**

```
generic
   type Source(<>) is limited private;
   type Target(<>) is limited private;
function Ada.Unchecked_Conversion(S : Source) return Target;
```

и применить результат настройки к нужным объектам (то есть, сделать такое случайно, «не заметив», невозможно).

Язык программирования и надежность – практика. Полезные «мелочи»…

- **Все намерения программиста должны быть явно обозначены**

```
type Week_Day is
 (Mon, Tue, Wen, Thu, Fri,
  Sat, Sun);
Day : Week_Day;
...
case Day is
   when Mon .. Thu =>
      Working_Like_Dog;
   when Sat =>
      Drinking;
   when Sun =>
      Fishing;
end case;
-- Но что происходит в пятницу???
-- Такой код не будет скомпилирован!
```

- **Статическая борьба с «дохлыми» ссылками:**

```
package P is
   type T is ...;
   type T_Ptr is access all T;
   Global : T_Ptr;
end P;
...
procedure Q is
   X : aliased T;
   Local : T_Ptr := X'Access; --  Так нельзя!
   --  Local может быть присвоен чему угодно,
   --  в т.ч. и  Global
begin
   Global := X'Access; -- Нельзя!
   Global := Local; -- Нельзя!
   Global := X’Unchecked_Access;
   --  Можно, но под личную
   --  ответственность автора
...
end Q;
```

Язык программирования и надежность – практика. Подтипы и исключительные ситуации

- **Тип данных – это:**
  - концептуально - содержательная роль объектов данных в программе;
  - технически – множество значений + совокупность операций;
- **Подтип – это ограничение множества значений в рамках той же содержательной роли;**
- **Реализация сама проверяет, что присваиваемое значение попадает в подтип, определенный для данного объекта данных (принадлежит подтипу);**
- **В случае нарушения подтипового ограничения при выполнении программы возбуждается предопределенное ограничение Constraint_Error;**

Язык программирования и надежность – практика. Подтипы - пример

```
procedure Автопилот is
   type Скорость is digits 8 range 0.0 .. 500.0;
   function Данные_Спидометра return Скорость is ... end Данные_Спидометра;
   subtype Безопасная_Скорость is Скорость range 0.0 .. 200.0;
   Текущая_Скорость : Безопасная_Скорость;
   ...
begin
   ...
   loop
      Текущая_Скорость := Данные_Спидометра;
      ...
   end loop;
   ...
exception
   when Constraint_Error =>
      Включить_Аварийный_Сигнал;
      Нажать_На_Тормоз;
      ...
end Автопилот;
```

Язык программирования и надежность – практика. Подтипы и исключительные ситуации

- **Для одного типа данных можно определить сколько угодно разных подтипов;**

- **Подтиповые ограничения формулируются статически, но определяются и проверяются – динамически**

```
subtype Безопасная_Скорость is Скорость
    range Min (Режим_Полета) .. Max (Режим_Полета);
```

- **Подтипы можно применять при конструировании сложных структур данных;**

- **Существуют различные виды ограничений;**
- **Подтиповые ограничения всегда проверяются динамически;**

Язык программирования и надежность – практика. Проблемы при сборке программы

```
package P1 is
   function F1 return Integer;
end P1;
```

```
package P2 is
   function F2 return Integer;
end P2;
```

```
with P2;
package body P1 is
   X1 : Integer := P2.F2;
   function F1 return Integer is
   begin
      return X1;
   end F1;
end P1;
```

```
with P1;
package body P2 is
   X2 : Integer := P1.F1;
   function F2 return Integer is
   begin
      return X2;
   end F2;
end P2;
```

Реализация Ады обязана определить, что не существует корректного порядка включения кода, порождаемого модулями P1 и P2, в исполняемый код, а потому никогда не будет создан исполняемый код для программы, в состав которой входят эти модули.

На обеспечение надежности работают и такие языковые механизмы, как:

- **Разделение спецификации (интерфейса) и реализации (тела) для каждого программного модуля и невозможность доступа к телам со стороны клиентов модуля;**
- **Приватные типы;**
- **Правила видимости имен;**
- **Правила раздельной компиляции;**
- **Контрактная модель настройки шаблонов;**
- **…**

Чего стоят такие меры обеспечения надежности?

- Статический контроль (как при раздельной компиляции модулей, так и при сборке программ) приводит только лишь к утяжелению транслятора, никак не сказываясь на размерах и быстродействии исполняемого кода;

- Динамический контроль (подтиповые ограничения) незначительно утяжеляют исполняемый код, при необходимости они могут быть (выборочно) отключены;

- Зато шансов, что до исполнения будет допущен код с «дурными» ошибками, которые к тому же не будут сразу обнаружены, намного меньше!

И что они дают?

IEEE Software, 1987, #1, pp40-44:

- **производительность разработчиков на полном цикле (>= 20_000 SLOC):**
  - Ада - 12 строк/день
  - другие языки - 10 строк/день

- **распределение затрат в проекте:**

	проекти-рование	реализация	отладка
Ада	64.4	27.1	8.3
другие языки	28-40	17-58	24-50

Асинхронные процессы в Аде

- **Наш мир состоит из взаимодействующих асинхронных процессов, взаимодействующих друг с другом и конкурирующих за использование разделяемых ресурсов;**
- **Программы, встроенные в реальный мир, должны уметь работать с асинхронными процессами и разделяемыми ресурсами;**
- **Программы с асинхронными процессами существенно сложнее последовательных программ;**
- **Чем может помочь язык программирования?**
  - Предоставление высокоуровневых средств описания асинхронных процессов и управления ими;
  - Данные средства должны быть согласованы с привычными нам моделями взаимодействия асинхронных процессов;
  - Данные средства должны быть ориентированы на устойчивые к ошибкам парадигмы и технологии работы с асинхронными процессами;
- **Ада – единственный индустриальный язык, который это умеет.**

Два асинхронных процесса на 20 строк кода (by B. Brosgol, http://www.embedded.com/story/OEG20021211S0034)

```
1  with Ada.Text_IO;
2  procedure Tasking_Example is
3     Finished : Boolean := False;
4     pragma Atomic(Finished);
5
6     task Outputter;   -- task specification
7
8     task body Outputter is   -- task body
9        Count : Integer := 1;
10    begin
11       while not Finished loop
12          Ada.Text_IO.Put_Line(Integer'Image(Count));
13          Count := Count + 1;
14          delay 1.0;   -- one second
15       end loop;
16       Ada.Text_IO.Put_Line("Terminating");
17    end Outputter;
18
19 begin
20    delay 20.0;   -- twenty seconds
21    Finished := True;
22 end Tasking_Example;
```

процесс Tasking_Example

процесс Outputter

результат работы этой программы:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
Terminating
```

«Умный» буфер

```
protected type Bounded_Buffer is
   entry Put (X:  in Item);
   entry Get (X: out Item);
private
  A      : Item_Array (1 .. Max);
  I, J   : Integer range 1 .. Max := 1;
  Count  : Integer range 0 .. Max := 0;
end Bounded_Buffer;
```

```
--  Использование буфера:
--  * возможно только в режиме
--    взаимного исключения;
--  * тупики исключены;
    My_Buffer : Bounded_Buffer;
    ...
    My_Buffer.Put (Something);
    ...
    My_Buffer.Get (To_Something);
```

```
protected body Bounded_Buffer is

   entry Put (X : in Item)
      when Count < Max
   is
   begin
      A (I)   := X;
      I       := I mod Max + 1;
      Count   := Count + 1;
   end Put;

   entry Get (X : out Item)
      when Count > 0
   is
   begin
      X     := A(J);
      J     := J mod Max + 1;
      Count := Count - 1;
   end Get;

end Bounded_Buffer;
```

Почему Ада оказывается эффективным решением для образования?

- **По тем же самым причинам, по которым Ада – эффективное решение для индустрии:**
  - язык построен систематическим образом с опором на целостную философию, ключевым моментом которой является надежность программных услуг;
  - предоставляя больше возможностей, язык оказывается проще в изучении и использовании, обладая к тому же ясным и легко читаемым синтаксисом;
- **Возможность (бесплатного) использования в учебном процессе индустриальной Ада-технологии (GNAT Academic Program);**
- **Простота адаптации учебных программ и курсов, основанных на Паскале.**

Хотите узнать больше:

- **http://www.ada-ru.org/index.html**
  - все по-русски, включая книгу по языку Ада-95;
  - много полезных ссылок;
- **http://www.adacore.com**
  - помимо описания GNAT-технологии, богатый набор справочных материалов по языку и практических советов по его использованию;
- **http://www.adaic.com**
  - стандарт Ады (все редакции + версии с комментариями), сопутствующие стандарты (ASIS), обоснование проектных решений, ссылки на учебники;