Александр Теленьга, *старший преподаватель кафедры информационных систем факультета прикладной информатики Института экономики, права и гуманитарных специальностей, кандидат педагогических наук, г. Краснодар*

РОЛЕВОЕ ИНФОРМАЦИОННОЕ МОДЕЛИРОВАНИЕ В ОБУЧЕНИИ ПРОГРАММИРОВАНИЮ СТУДЕНТОВ МЛАДШИХ КУРСОВ ВУЗОВ

Под ролевым информационным моделированием (РИМ) мы понимаем такое моделирование, при котором постановка задачи на разработку модели и/или её последующий анализ осуществляются с точек зрения людей, выступающих в различных социальных ролях. Такая методология требует от студентов умения определять проблемы и ставить задачи, выделять главные и второстепенные свойства объектов, необходимые для построения моделей, уметь решать поставленные задачи с использованием различных инструментов, в том числе и программных средств, прогнозировать и оценивать результаты своей работы[1].

При обучении программированию студенты зачастую выступают в одной роли — роли ***разработчика***, и процесс изучения языка программирования строится исходя из этой позиции. При таком подходе некоторые конструкции языка (например, комментарии), на первый взгляд, кажутся избыточными и не требующими внимания. Методика РИМ позволяет по-новому раскрыть назначение отдельных инструментов языка программирования, подчеркнуть их важность для грамотного решения поставленной задачи. Голландский учёный Эдсгер В. Дейкстра отметил: «Глубоко ошибается тот, кто думает, что изделиями программистов являются программы, которые они пишут. Программист обязан создавать заслуживающие доверия решения и представлять их в форме убедительных доводов, а текст написанной программы является лишь сопроводительным материалом, к которому эти доказательства применимы».

Проиллюстрируем вышесказанное примером. Необходимо написать программу вычисления двойного факториала для нечёт-

[1] См.: *Теленьга А.П.* Ролевое информационное моделирование в обучении компьютерным телекоммуникациям в профессиональной подготовке студентов экономических специальностей вузов: Дисс. ... канд. пед. наук. Краснодар, 2009; *Юнов С.В.* Ролевое информационное моделирование в педагогической деятельности, 2010.

ного числа ***n***, используя рекурсию. Напомним, что двойной факториал является произведением всех натуральных чисел той же чётности, что и ***n***, до ***n*** включительно.

Одним из вариантов решения этой задачи будет следующая программа на языке Pascal:

```
program oddfactorial;
uses crt;
var n: integer;
function F(n: integer): longint;
begin
if n = 1 then F:= 1
else
if n = 3 then F:= 3
else F:= n*F(n-2);
end;
begin
ClrScr;
repeat
    Readln(n);
until odd(n);
Write(F(n));
Readln;
end.
```

С точки зрения разработчика, эта программа полностью решает поставленную задачу. Однако если мы проанализируем решение с позиции ***пользователя***, то предложенный вариант будет нуждаться в доработке.

Действительно, если мы откомпилируем программу и запустим её на выполнение, то для пользователя процесс работы с ней будет похож на общение с «чёрным ящиком»: программа на входе считывает числа до тех пор, пока не будет введено нечётное, и на выходе выдаёт вычисленный ответ. Естественно, что ввод и вывод программы должны содержать инструкции для пользователя:

```
program oddfactorial;
uses crt;
var n: integer;
function F(n: integer): longint;
begin
if n = 1 then F:= 1
else
if n = 3 then F:= 3
else F:= n*F(n-2);
end;
begin
ClrScr;
repeat
    Write ('Введите, пожалуйста, нечётное число для вычисления двойного факториала: ');
    Readln(n);
until odd(n);
Write ('Двойной факториал для числа ',n,' равен ');
Write(F(n));
Readln;
end.
```

Таким образом, роль ***пользователя*** помогает нам акцентировать внимание на грамотном оформлении ввода и вывода данных, что не всегда очевидно для студентов. На введение этой роли обращали внимание С.А. Бешенков и Е.А. Ракитина[2].

Рассмотрим ещё одну роль — ***стороннего разработчика***. Её необходимость вызвана тем, что крупные проекты требуют координированных действий большого числа разработчиков. Для стороннего разработчика важнейшим требованием к программе является её понятность. Достичь этого можно, соблюдая несколько стилей программирования,

[2] См.: *Бешенков С.А., Ракитина Е.А.* Моделирование и формализация. Методическое пособие. М.: Лаборатория базовых знаний, 2002.

которые не очевидны для студентов младших курсов.

Концепция первого стиля — в использовании понятных имён переменных. Согласно венгерской нотации, например, в имени переменной должна содержаться информация о ней, прежде всего о её типе: имя переменной pszLicFileContents говорит о том, что она указатель (p), на строку фиксированного размера (sz), в названии переменной pbContent закодирован её тип LPBYTE, переменная bVerified имеет тип bool. В настоящее время этот подход считается не таким удобным, прежде всего тем, что затрудняет чтение кода. Строгая типизация не может быть эффективно заменена искусственными средствами, и поддержание этих искусственных средств не оправдывает усилий. Современные подходы предлагают называть переменные исключительно по смыслу: SelectedIndexInTable, OpenedFileName.

Оформление исходного кода программы также позволяет повысить её ясность. Язык программирования Pascal допускает перечисление операторов программы в одну строчку с разделителями «;» и «.», однако очевидно, что такая запись непонятна и сложна для восприятия. В указанных нами примерах блоки определения переменных, циклы, операторы ввода и вывода оформлены отступами. Существуют языки программирования, например, Python, которые не допускают запуска программы без правильного оформления кода.

Кроме того, значительно повысить ясность кода помогают комментарии. Рекомендуется подробно комментировать каждый объект и все публичные методы объекта программы или модуля, а также переменные, если таковые есть, плюс обязательны комментарии для внутренних методов и переменных, а также внутри функций необходимо комментировать особо сложные и важные моменты. Существует подход, предлагающий прежде чем писать функцию, описывать её поведение подробно в комментарии:

```
{Функция getlines считывает введённые с клавиатуры строки, записывает их в массив str и возвращает количество считанных строк}
function getlines(var str: array of string): integer;
var
i: integer; {Счётчик строк}
begin
i:= 0; {Инициализация счётчика}
{Пока не достигнут конец файла, считываем следующую строку и увеличиваем счётчик}
while not eof (input) do
begin
    Readln(str[i]);
    inc(i);
end;
{Возвращаем количество считанных строк}
getlines:= i;
end;
```

У сторонников комментирования исходного кода есть противники, утверждающие, что комментарии подспудно подталкивают разработчика писать непонятный код (ведь в комментариях есть разъяснение). Об этом говорилось, например, в статье С.В. Юнова «О принципе историзма в обучении информатике»[3]. Это утверждение весьма разумно, однако, поскольку уровень подготовки разработчиков различен, оптимальным выходом из

[3] *Юнов С.В.* О принципе историзма в обучении информатике // Информатика и образование. 2009. № 1.

сложившейся дилеммы будет симбиоз понятного кода и комментариев для наиболее сложных участков программы.

Следуя перечисленным выше подходам, текст программы из предложенного нами примера должен быть модифицирован следующим образом:

```
program oddfactorial;
uses crt;
var n: integer;
{Функция DoubleFactorial считает двойной факториал от переданного ей числа n}
function DoubleFactorial(n: integer): longint;
begin
{Первые два условия взяты из определения двойного факториала}
if n = 1 then DoubleFactorial:= 1
else
if n = 3 then DoubleFactorial:= 3
{Рекурсивный вызов функции}
else DoubleFactorial:= n*DoubleFactorial(n-2);
end;
begin
{Перед началом работы очищаем экран}
ClrScr;
{Ввод чисел продолжается до тех пор, пока не будет введено нечётное число}
repeat
Write ('Введите, пожалуйста, нечётное число для вычисления двойного факториала: ');
Readln(n);
until odd(n);
Write ('Двойной факториал для числа ',n,' равен ');
{В операторе Write вызываем функцию DoubleFactorial}
Write (DoubleFactorial(n));
Readln;
end.
```

Таким образом, использование ролевого информационного моделирования при обучении программированию студентов младших курсов позволяет выявить возможные проблемы уже на этапе разработки модели решения задачи, шире взглянуть на возможности конструкций языка программирования, акцентировать внимание будущих разработчиков на грамотном оформлении кода программы и заботе о пользователе.