УДК 004.514.62
DOI: 10.25559/SITITO.18.202203.596-607

Оригинальная статья

Онтология практических знаний по программированию

А. П. Гагарин
ФГБОУ ВО «Московский авиационный институт (национальный исследовательский университет)», г. Москва, Российская Федерация
Адрес: 125993, Российская Федерация, г. Москва, Волоколамское шоссе, д. 4
gagarin_ay@outlook.com

Аннотация

Стремительное развитие информационных технологий, изменчивость методических установок в сфере образования ведут к хронической нестабильности комплекса учебно-методических материалов в практике преподавания программирования и смежных с ним дисциплин. Профессиональный преподаватель сталкивается с необходимостью оперативно изменять структуру и содержание лекций, заданий для лабораторных и семинарских занятий. Формируется объективная потребность в методах и инструментах накопления и специализированного редактирования учебных материалов, среди которых особую роль играют учебные примеры программ. Особенно ценны примеры, представляющие собой тексты программ на исходных языках программирования, пригодные для выполнения в соответствующей операционной среде с выдачей протокола выполнения, который, по существу, можно считать входящим в состав примера. Целесообразно накапливать коллекции таких примеров и включать их с большими или меньшими изменениями в учебные пособия, доклады и демонстрационные материалы. В данной статье коллекции примеров рассматриваются как контент баз практических знаний, разновидность «больших данных», необходимых для конструирования исходных текстов программ на языках программирования. Предложена архитектура программной оболочки комплексной базы знаний, объединяющей прагматику программирования с синтаксисом и семантикой языков программирования. В оболочку встроены фрагмент онтологии языка Java и фрагмент онтологии умений («компетенций») программиста, необходимых для реализации ряда программных решений и выраженных в виде примеров на языке Java. Комплексная база знаний позволяет в среде Windows выбирать и демонстрировать примеры одновременно с демонстрацией соответствующих определений из спецификации языка Java.

Ключевые слова: программирование, язык программирования, синтаксис, семантика, прагматика, синтаксическая прагматика, семантическая прагматика, вычислительная прагматика, онтология, учебный пример, шаблон

Конфликт интересов: автор заявляет об отсутствии конфликта интересов.

Для цитирования: Гагарин А. П. Онтология практических знаний по программированию // Современные информационные технологии и ИТ-образование. 2022. Т. 18, № 3. С. 596-607. doi: https://doi.org/10.25559/SITITO.18.202203.596-607

Original article

Ontology of Pragmatic Knowledges on Program Implementation

A. P. Gagarin
Moscow Aviation Institute (National Research University), Moscow, Russian Federation
Address: 4 Volokolamskoe shosse, Moscow 125993, Russian Federation
gagarin_ay@outlook.com

Abstract

Swift development of the information technologies as well as a notable variability of methodological settings in the professional education entail a chronic volatility of the hole complex of didactic guides and study manuals in the current practice of teaching programming and related disciplines. Professional teacher faces the need to change promptly the structure and content of her/his lectures, laboratory works and practical lessons. An objective demand emerges on methods and tools for saving and editing of didactic materials, that include didactic examples of programs. The text of a source program is considered highly appropriate for this purpose if it is successfully executable in a suited operational environment and provides snapshots of the result. The snapshots may be considered as parts of the examples. It is very practical to collect such examples and include them as they are or with small edits into the current manuals, reports and presentations. Collections of such examples are regarded in the issue as a content of databases, a sort of the "big data", that accumulates pragmatic knowledges needed for constructing source texts in programming languages. An architecture of a knowledge base shell is proposed that integrates the pragmatics of program constructing with syntax and semantics of the programming language in use. The shell includes a piece of a Java language ontology and a piece of an ontology of programmer's skills (competences) needed for implementation of a set of solutions and expressed as Java-examples. The integrated knowledge base lets to select and display in the Windows-environment the amassed program examples in parallel with showing definitions taken from the specification of Java.

Keywords: program implementation, programming language, syntax, semantics, pragmatics, syntactic pragmatics, semantic pragmatics, computational pragmatics, ontology, didactic example, pattern

Conflict of interests: The author declares no conflict of interest.

For citation: Gagarin A.P. Ontology of Pragmatic Knowledges on Program Implementation. *Modern Information Technologies and IT-Education.* 2022;18(3):596-607. doi: https://doi.org/10.25559/SITITO.18.202203.596-607

Введение

Обучение на примерах, наряду с обучением методом проб и ошибок, представляется древнейшим и часто эффективнейшим способом обучения, корни которого, видимо, кроются во врождённых неосознанных или слабо осознанных реакциях подражания. Умение «писать» программы на языке программирования приобретается, в частности, гораздо быстрее, если обучающийся имеет возможность подражать имеющимся образцам, копируя их в точности или несущественно адаптируя их к цели речевого акта, каковым, по существу, является любая составленная человеком компьютерная программа. Как индивидуальный опыт поколений программистов, так и общественный опыт, отражённый в образовательных методиках, свидетельствует, что совокупность программистских профессиональных умений входит в состав особого вида знаний – прагматического, отличающегося от знания синтаксиса и семантики языков программирования.

В той мере, в какой объективными носителями синтаксического знания выступают спецификации языков программирования, платформ («фреймворков»), носителями прагматического знания являются тексты программ, выделенные как примеры, образцы, или шаблоны («patterns»). Семантические знания языков программирования фиксируются в законченной форме в формальных спецификациях семантики (если таковые имеются) и, в общем случае, – частично в примерах, частично в пояснениях к формализованным спецификациям синтаксиса языков.

Программная инженерия преподаётся на примерах. В частности, в пяти взятых случайно популярных учебных пособиях по языку Java, в частности, в работе[1], один пример приходится в среднем на 2-3 страницы. Таким образом, пример в учебном пособии – явление заурядное и при всей заурядности – совершенно необходимое. Недостаточность примеров в составе учебного курса оценивается как весомая причина слабой усвояемости этого курса обучающимися [1]. Важную роль в обучении играют также примеры-«задачи», решение которых выражается в «доработке» обучающимся в целях проверки и закрепления программистских знаний и навыков [2]. Документация продуктов Майкрософт изобилует примерами, распределёнными по продуктам, компонентам и их функциям.

Знания по программной инженерии имеют в большей своей части вид шаблонов. Профессиональные стандарты, например[2], требуют от программистов не столько знаний, сколько умений. Правила и «законы» являются, по существу своему, обобщением шаблонов, архитектурных и конструктивных. Стратегия преподавания заключается в выборе путей от номенклатуры умений к примерам, показывающим из чего эти умения складываются и как проявляются, а также демонстрирующим необходимые правила (хотя встречается навигация и в обратном направлении) [3].

Несмотря на видное место, которое примеры, как тексты занимают в пространстве учебной и эксплуатационной документации, а также по продолжительности пользования ею, они редко являются предметом научного исследования. В данной статье предпринята попытка выработать принципы таксономии примеров, типологию их использования, требования к различным категориям примеров, способы идентификации примеров при хранении и поиске.

Уточнение понятия «пример программы»

Как нередко бывает в случае выработки строгого определения для понятия, применяемого широко, но обделённого вниманием теории, определение «примера программы» представляет собой заметную проблему. Русскоязычные словари: Викисловарь[3], словарь Ожегова[4], Энциклопедический словарь[5] – толкуют «пример» двояко: как частный случай (экземпляр), оцениваемый нейтрально по отношению к свойствам подлинника, и как образец, наглядный, поясняющий, типичный, заслуживающий подражания, то есть с позитивным окрашиванием подлинника и/или указанием как источника дополнительных сведений.

В англоязычном словаре Merriam-Webster[6] «example» и «sample» (как аналоги русскоязычной лексемы «пример») истолковываются, как и в указанных выше словарях, но с другой очерёдностью вариантов толкования, кроме того «sample» – не вносит оценочной окраски. Зато имеется аналог с оценочной окраской – «specimen».

Поиск в сети Интернет показывает, что обе лексемы, и «пример», и «пример программы» не имеют самостоятельного технического значения. Они не являются терминами.

Ближайшим обобщающим толкованием для словосочетания «пример программы» предлагается принять «текст на одном или нескольких языках программирования, возможно, с комментариями на естественном языке». Несомненным характеристическим свойством такого примера является наличие «темы», которую «раскрывает» или «демонстрирует» пример. Вторым характеристическим свойством этого текста является его пригодность быть отображённым хотя бы в одну исполняемую программу так, что эта программа сама была бы примером на ту же тему. То есть, иными словами, пример – это элемент множество текстов, демонстрирующих некоторую тему. Пример программы не обязан быть и обычно не является программным продуктом – изделием программной промыш-

[1] Дубаков А. А. Введение в объектно-ориентированное программирование на Java : учеб. пособие. СПб : Университет ИТМО, 2016. 250 с. URL: https://books.ifmo.ru/file/pdf/2139.pdf (дата обращения: 17.08.2022).

[2] Об утверждении профессионального стандарта «Системный программист» : приказ Министерства труда и социальной защиты Российской Федерации от 29.09.2020 № 678н [Электронный ресурс]. URL: https://docs.cntd.ru/document/566113547?marker=6500IL (дата обращения: 17.08.2022).

[3] Пример [Электронный ресурс] // Викисловарь, 2022. URL: https://ru.wiktionary.org/wiki/пример (дата обращения: 17.08.2022).

[4] Пример [Электронный ресурс] // Толковый словарь Ожегова онлайн, 2017. URL: https://slovarozhegova.ru/word.php?wordid=23877 (дата обращения: 17.08.2022).

[5] Пример [Электронный ресурс] // Энциклопедический словарь, 2012. URL: https://slovar.cc/enc/slovar/1775181.html (дата обращения: 17.08.2022).

[6] Merriam-Webster Dictionary : официальный сайт компании Merriam-Webster, Incorporated [Электронный ресурс]. URL: https://www.merriam-webster.com (дата обращения: 17.08.2022).

ленности.

Частными случаями темы являются определения из спецификации языка программирования, правила размещения строк программы на листе (отступы и т.п.), такие понятия информатики как «синхронизация», «обработка особого случая». Попытка более общего, но при том конструктивного определения темы приводит к признанию того, что темой может выступать элемент любой модели обработки информации, определённой в контексте применения примера.

Дополнительными (не-характеристическими и не обязательными) свойствами примера программы являются законченность и исполнимость. Законченность проверяема, если пример состоит из элементов спецификации, образующих конструкцию, законченную в смысле этой спецификации. Исполнимость состоит в том, что для примера можно указать операционную среду, в которой пример может быть выполнен как приложение или сервис. Неисполнимый пример может быть потенциально исполнимым, если его можно дополнить, обеспечив исполнимость. Учебный пример-задача должен быть потенциально исполнимым. Обратно, пример можно минимизировать, сокращая текст с потерей или без потери исполнимости. В результате для любого примера можно указать его тематическое «ядро». Пример, неисполнимый из-за текстуальных особенностей, неустранимых путём дополнения, считается «примером с ошибкой». Такие примеры должны содержать соответствующие комментарии.

Пример может быть многотемным. Темы могут быть разделимыми, если путём сокращения текста можно выделить «одно-темное» ядро. Встречаются ядра, связанные с одной или более темами, которые уже не поддаются разделению.

На Рис. 1 и 2 показан исполнимый пример на тему «Шаблон generic для классов, свойств и событий в языке C++», состоящий из текста на языке C++ в языковой проекции CLI и протокола исполнения. Пример взят из преподавательской практики автора.

Пример является многотемным. Первая тема непосредственно демонстрируется участком программы, который выделен комментариями вида **** Опыт I ****. На этом участке создаются объекты класса, определённого с участием типа-параметра. Для первого объекта в тип-параметр подставляется тип int, во второй объект – тип double. В контексте этих объектов производится обращение к объектам-данным, свойствам (property) и вызов метода.

Участки программы «Опыт II», «Опыт Ш» и «Опыт IV» демонстрируют остальные 3 темы по работе с самореализующимся свойством, делегатами и событиями. Каждый из названных участков примера может использоваться как самостоятельный исполнимый пример при условии наличия в пространстве имён определений классов A и B и делегата Legat.

На Рис. 2 приведён протокол исполнения рассмотренного примера.

На Рис. 3 приведён исполнимый пример на тему создания элемента класса (Example 12.5-1. Evaluation of Instance Creation из спецификации языка Java[7]).

Понятие «образец», или «шаблон» (в англоязычной литературе «pattern») укладывается в понятие «пример программы», определённое в данной статье. Для образца характерна высокая степень обобщённости темы. Классический пример образцов программ – образцы GOF[8] [4].

Проблемы применения примеров программ

Примеры программ, рассматриваемые в статье, встречаются не только в учебных пособиях, предназначенных для первичного освоения языка программирования в сфере специального образования, но и в научно-технической литературе, отслеживающей развитие языков и предназначенной, главным образом, для опытных программистов (например, сборник А. Васильева[9]), а также в спецификациях языков, библиотек, расширяющих языки, и фреймворков.

В работе [5] спецификация языка программирования, ещё называемая определением или стандартом, трактуется как документ, определяющий язык программирования так, чтобы пользователи и разработчики языка могли прийти к согласованному мнению, что означают программы на данном языке[10]. Спецификация может иметь не нормативный, а авторский характер, выражая точку зрения для последующего обсуждения. В статье используются спецификации языков C++ и Java[11], согласованное понимание при чтении каждой из которых достигается явным определением синтаксиса и семантики специфицируемого языка. В других случаях предъявляется эталонная реализация на специфицируемом языка, то есть, фактически, «мега-пример».

Синтаксис языка программирования обычно описывается с использованием комбинации формализма регулярных выражений и контекстно-свободной грамматики, а семантика излагается неформально, на естественном языке [6-10]. Для отдельных языков программирования, например, Algol-60 [11], C[12], формализация семантики проведена, но формальные

[7] The Java® Language Specification / J. Gosling [и др.]. Java SE 19 Edition. Oracle America, Inc.; 2022. [Электронный ресурс]. URL: https://docs.oracle.com/javase/specs/jls/se19/html/index.html (дата обращения: 17.08.2022).

[8] Gangs of Four (GoF) Design Patterns [Электронный ресурс] // DigitalOcean, 2022. URL: https://www.digitalocean.com/community/tutorials/gangs-of-four-gof-design-patterns (дата обращения: 17.08.2022). Design Patterns: Elements of Reusable Object-Oriented Software / E. Gamma, R. Helm, R. Johnson, J. Vlissides. 1st Edition. Addison-Wesley Professional, 1994. 416 p.

[9] Васильев А. Н. Программирование на C++ в примерах и задачах. М.: ЭКСМО, 2020. 368 с.

[10] Gunter C. A. Semantics of Programming Languages: Structures and Techniques. MIT Press, Cambridge, MA; 1992. 441 p.

[11] Programming Languages – C++. ISO/IEC JTC1 SC22 WG21 N 4860. ISO; 2020. [Электронный ресурс]. URL: https://isocpp.org/files/papers/N4860.pdf (дата обращения: 17.08.2022); JTC1/SC22/WG21 – The C++ Standards Committee – ISOCPP [Электронный ресурс]. URL: https://www.open-std.org/JTC1/SC22/WG21 (дата обращения: 17.08.2022); The Java® Language Specification / J. Gosling [и др.]. Java SE 19 Edition. Oracle America, Inc.; 2022. [Электронный ресурс]. URL: https://docs.oracle.com/javase/specs/jls/se19/html/index.html (дата обращения: 17.08.2022).

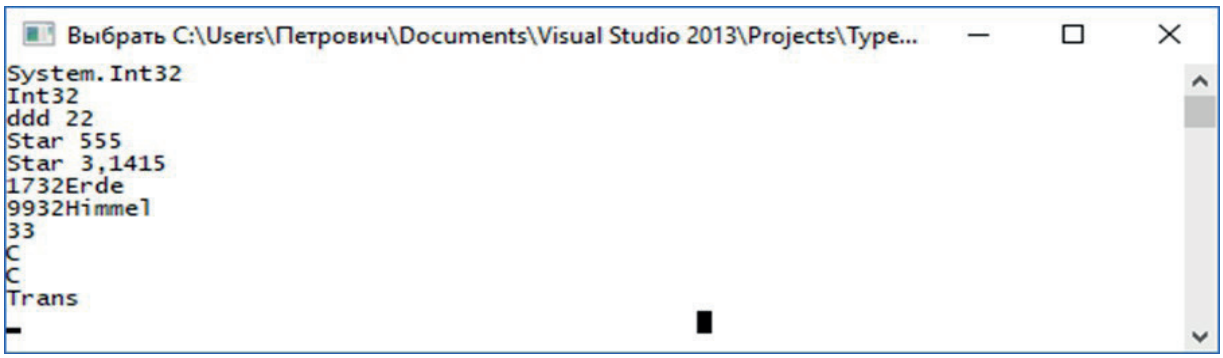
[12] Papaspyrou N. A Formal Semantics for the C Programming Language : Electronic Theses and Dissertations. Athens : National Technical University of Athens, 1998. 253 p. URL: http://www.softlab.ntua.gr/~nickie/Papers/papaspyrou-1998-fscpl.pdf (дата обращения: 17.08.2022).

```
#include "stdafx.h"
#include "pch.h"
using namespace System;
ref class C;
generic <typename P> public delegate P Legat(P p) ;//для опыта IV
generic <class T, typename Q> ref class A
{
public: A(T s) { X = s; }
public:  T X;
public: void F()
        {
                Console::WriteLine(X->GetType());
                Console::WriteLine(X->GetType()->Name);
                Console::WriteLine("ddd {0}", X);
        }
public: Q Root;
public: property T Star;
public: delegate T Del(T t, String ^ss);//для опыта III
public: Del ^dd;//для опыта III
public: Q Trans(Q q) { Console::WriteLine("Trans"); return q; }//для опыта IV
                static Legat<C^>^ lgt;//для опыта IV
};
public  ref class C
{
        String^Body;
 public: C ^Next;
 public: int Join(int n, String^sent)
 {
        Console::WriteLine(n.ToString() + ' ' + sent);
        return n;
 }
};
generic <typename S> Boolean Check(S s) { Console::WriteLine(s->ToString()); return true; }//для опыта IV

int main(array<System::String ^> ^args)
{
        //**********************  Опыт I ****************************
//      C ^c = gcnew C();
        A<int, C^> ^a = gcnew A<int, C^>(22);
   a->F();
        a->X = 11;
        a->Star = 555;
        Console::WriteLine("Star {0}", a->Star);
        A<double, C^> ^b = gcnew A<double, C^>(3.1415);
        b->X = 3.1415;
        Console::WriteLine("Star {0}", b->X);
        //**********************  Опыт II ****************************
        C ^d;
        d = gcnew C();
        a->Root = d;
        a->Root->Join(17, "Erde");
        //**********************  Опыт III ****************************
        a->dd = gcnew A<int, C^>::Del(d, &C::Join);
        a->dd(99, "Himmel");
        //**********************  Опыт IV ****************************
        Check(33);
        Check<C^>(d);
        Check(d);
        a->lgt = gcnew ::Legat<C^>(a, &A<int, C^>::Trans);
        a->lgt(d);
        Console::ReadKey();
        return 0;
}
```

Р и с. 1. Исходный текст примера на тему «Шаблон generic для классов, свойств и событий в языке C#»

F i g. 1. The source text of the example on the topic "Generic Template for Classes, Properties and Events in the C# Language"



Р и с. 2. Протокол исполнения текста программы примера
F i g. 2. Protocol for executing the text of the example program

```
class Point {
      int x, y;
      Point() { x = 1; y = 1; }
  }
class ColoredPoint extends Point {
      int color = 0xFF00FF;
  }
class Test {
      public static void main(String[] args) {
      ColoredPoint cp = new ColoredPoint();
      System.out.println(cp.color);
   }
}
```

Р и с. 3. Исходный текст примера на тему «Создание элемента класса»
F i g. 3. The source text of the example on the topic "Creating a Class Element"

семантики трудно назвать рабочим инструментом программиста [12]. Примеры в спецификации способствуют разъяснению семантики языка, однако их применение не имеет систематического обязательного характера, затрагивая лишь часть объектов спецификации.

Различаются два сценария применения примера программы:

- пример дополняет спецификацию языка, входя в неё или в отдельный документ (сборник примеров или учебное пособие),
- пример используется для конструирования подобных ему программных текстов в учебных,
- просветительных или производственных целях.

Второй сценарий может быть продолжением первого.

Подводя итог, позволительно сделать вывод, что примеры программ образуют своеобразный класс программных текстов, циркулирующих в публичном информационном пространстве, накапливающихся в персональном информационном пространстве программистов и играющих немаловажную роль среды распространения знаний практического программирования. Нельзя не отметить, что в настоящее время отсутствует общедоступная таксономия примеров, достаточно глубокая для их эффективного коллекционирования и выборки из коллекций в целях дальнейшего использования. Создание таких таксономий и соответствующих программных инструментов представляется возможным в виду наличия онтологий фундаментального характера, таких, как DOLCE [13], онтологий по языкам, как, например, Базы данных по лексике английского языка[13] [14, 15] и, в частности, онтологий по обучению программированию на языке C [16], обучению языку Java [17] и языкам Java и C++ [18].

В этой связи закономерно поставить вопрос, чем являются примеры в системе теоретического программирования.

Семантика и прагматика примеров программ

Вполне уместной отправной точкой для теоретического анализа понятия «пример программы» представляется обращение к понятию «язык программирования». По терминологическому стандарту ИСО в области систем и программной инженерии[14] язык программирования – это «язык, используемый для выражения компьютерных программ». В другом терми-

[13] WordNet: A Lexical Database for English : Princeton University website [Электронный ресурс]. URL: https://wordnet.princeton.edu (дата обращения: 17.08.2022).

[14] ISO/IEC/IEEE International Standard – Systems and software engineering – Vocabulary. ISO/IEC/IEEE 24765:2010(E). IEEE Computer Society, 2010. 418 p. doi: https://doi.org/10.1109/IEEESTD.2010.5733835

нологическом стандарте ИСО[15], который распространяется на информационные технологии в целом, к этому определению добавлено, что этот язык является искусственным. В стандарте[16] компьютерная программа определяется, как «сочетание компьютерных команд и определений данных, что позволяет аппаратуре компьютера выполнять вычислительные или управляющие функции». В стандарте[17] компьютерной программе даётся несколько другое определение: она «является синтаксической единицей, выполняющей правила языка программирования и составленной из объявлений и операторов или команд, нужных для выполнения определённой функции, задачи или решения проблемы».

Все приведённые выше определения не дают явных оснований к сведению языка программирования к естественным или искусственным языкам общения между людьми. Компьютерная лингвистика как бы выводится за рамки классического языкознания. Эта неявная интенция подтверждается хождением в литературе по программированию триады понятий «синтаксис», «семантика» и «прагматика», применяемых и к языкам программирования, и к текстам программ, и восходящей к научной традиции Ч. Пирса и Ч. Морриса[18].

Синтаксис и семантика определяются в стандарте [17] в близком соответствии с их толкованием в классическом языкознании. В спецификации языка C++ семантика языка определяется в терминах абстрактной машины[19]. Стандарты[20] прагматику не определяют. Прагматика в словаре Merriam-Webster возводится к семиотике: прагматика – «ветвь семиотики, изучающая отношение между знаком или языковым выражением и их пользователем»[21].

В работе[22] толкование прагматики применительно к программированию задает «конкретизацию абстрактного вычислителя для данной вычислительной системы». Прагматика языка программирования подразделяется на синтаксическую и семантическую. К синтаксической прагматике относятся правила сокращения записи, например, операторы инкремента и декремента целочисленного входного параметра. Семантическая прагматика – это возможность «определения того, что в описании языка оставлено на усмотрение реализации или предписывается в качестве вариантов вычислений». Для реализации этой возможности программисту предоставляются специальные языковые средства – «прагматические комментарии»[23] [19].

Трудно не заметить, что в пределах обоих определений исчезает первичный семиотический смысл прагматики, как выражения отношения между языком (элементами языка) и пользователем языка. Решая применить средства синтаксической или семантической прагматики, программист переходит к локальному варианту исходного языка. В случае семантической прагматики, он выражает готовность получить при этом даже другой результат, установив, например, прагматические комментарии {$I+}, {$I-} в программе на языке Pascal. То есть, пользователь выбирает другой язык, а не вариант пользования языком.

В данной статье учитывается специфика примеров программ, заключающуюся в том, что у них, фактически, два адресата: компьютер, способный пример исполнить, и человек, разбирающий («читающий») пример как текст. Исполнение примера на компьютере, как уже отмечалось, не является обязательным, в то время как восприятие примера программистом выступает главным назначением примера. Поэтому предлагается рассматривать прагматику в «узком смысле», близком к смыслу синтаксической прагматики: варианты программы, получающиеся в результате прагматического выбора, должны быть эквивалентны по результату выполнения.

Следует отметить, что такое понимание прагматики оставляет свободу толкования эквивалентности программ. Например, выбор варианта программы можно считать или не считать чисто прагматическим в зависимости от того, считать ли эквивалентными результаты выполнения, полученные за различное время или с различным расходом ресурсов памяти.

Совокупность примеров на некотором языке выражает прагматику этого языка, показывает, в частности, наилучшие («поучительные») случаи его использования наряду с вариантами, демонстрирующими определённые опасности. Пояснения в учебниках и спецификациях языка освещают и семантику, и прагматику языка, но они делают это декларативно в то время, как примеры – предметно [20].

Разбирая пример, программист проверяет и закрепляет понимание спецификации языка, кроме того, как подробно рассмотрено автором в [21], у программиста развивается готовность к составлению собственных аналогичных программных текстов. При этом реализация этой готовности в некоторой степени теряет осознанный характер, не требуя внутренней отсылки к выученным правилам и положениям спецификаций. То

[15] ISO/IEC 2382:2015 Information technology – Vocabulary. ISO, 2022. URL: https://www.iso.org/standard/63598.html (дата обращения: 17.08.2022).

[16] ISO/IEC/IEEE International Standard – Systems and software engineering – Vocabulary. ISO/IEC/IEEE 24765:2010(E). IEEE Computer Society, 2010. 418 p. doi: https://doi.org/10.1109/IEEESTD.2010.5733835

[17] ISO/IEC 2382:2015 Information technology – Vocabulary. ISO, 2022. URL: https://www.iso.org/standard/63598.html (дата обращения: 17.08.2022).

[18] Morris's C. W. Foundations of the Theory of Signs // International Encyclopedia of Unified Science. Vol. 1, no. 2. University of Chicago Press, 1938. P. 1-59.

[19] Programming Languages – C++. ISO/IEC JTC1 SC22 WG21 N 4860. ISO; 2020. [Электронный ресурс]. URL: https://isocpp.org/files/papers/N4860.pdf (дата обращения: 17.08.2022); JTC1/SC22/WG21 – The C++ Standards Committee – ISOCPP [Электронный ресурс]. URL: https://www.open-std.org/JTC1/SC22/WG21 (дата обращения: 17.08.2022).

[20] ISO/IEC/IEEE International Standard – Systems and software engineering – Vocabulary. ISO/IEC/IEEE 24765:2010(E). IEEE Computer Society, 2010. 418 p. doi: https://doi.org/10.1109/IEEESTD.2010.5733835; ISO/IEC 2382:2015 Information technology – Vocabulary. ISO, 2022. URL: https://www.iso.org/standard/63598.html (дата обращения: 17.08.2022).

[21] Merriam-Webster Dictionary : официальный сайт компании Merriam-Webster, Incorporated [Электронный ресурс]. URL: https://www.merriam-webster.com (дата обращения: 17.08.2022).

[22] Непейвода Н. Н. Стили и методы программирования : учеб. пособие. М. : ИНТУИТ.РУ, 2012. 320 с.

[23] Городняя Л. В. Парадигмальный подход к факторизации определений языков и систем программирования // Системная информатика. 2018. № 12. С. 1-26. doi: https://doi.org/10.31144/si.2307-6410.2018.n12.p1-26

есть, формируется «навык», определяемый в общеязыковых и специальных словарях (например, в Энциклопедическом словаре по психологии и педагогике): как «действие, сформированное путем повторения, характеризующееся высокой степенью освоения и отсутствием поэлементной сознательной регуляции и контроля»[24]. Поскольку развёрнутый процесс программирования в полностью автоматической, бессознательной форме в практике не встречается, правильнее полагать, что формируется «умение», определяемое в Социально-педагогическом словаре, как «освоенный субъектом способ выполнения действия, обеспечиваемый совокупностью приобретенных знаний и навыков»[25].

Иными словами, предлагается, исходя из рассмотренной природы прагматики примеров программ, положить её в основу кластеризации примеров, стараясь не потерять при этом связи с их семантикой. Тогда пример программы будет квалифицировать по уменью, формированию которого этот пример способствует. Рекомендуется, чтобы тема примера выражала это уменье (на англ. – skill).

Следует отметить, что языкознание и психология активно изучают феномен прагматики как поведения человека и, в частности, его языкового поведения. В работах D. Jurafsky [22] и H. Bunt [23] изучается отношение между высказыванием, с одной стороны, и действием, речевым контекстом, местом, временем и средой, в которых высказывание случается, с другой стороны. Хотя информационные и компьютерные технологии, в том числе и специализированные, используются в этих исследованиях как вспомогательные факторы – как среда и инструмент моделирования, но исследования в целом квалифицируются как «вычислительная прагматика». Состояние этого направления исследований на начало 1916 года отражено в материалах семинара[26], а также в обзоре X. Li и других [24]. В последнем отмечается, что «вычислительная прагматика» развивается как часть лингвистики, использующая прагматику как теоретический базис, позволяющий описать динамику отношения между дискурсом и контекстом, пользуясь теориями и методами компьютерных наук и технологий. В прикладном плане «вычислительная прагматика» ставит задачи в интересах программной инженерии, в частности, задачу исследования нужд пользователей программных продуктов, а также задачу разработки более точных и дружественных программисту компиляторов. Сюда же относится попытка синтеза программ на основе прагматического анализа их спецификаций, рассмотренная в [25, 26].

Тематика данной статьи в своей основе далека от вычислительной прагматики, однако коллекции примеров программ могут быть интересны как корпуса прагматических текстов, а понимание программ может быть исследовано как критерий прагматического выбора.

Онтологии языков программирования и прагматические знания

Рассмотрение примеров программ показывает, что небольшие примеры, восходящие к спецификациям языков и представляющие приёмы работы с элементами языка, образуют группы, ассоциированные с этими элементами. Например, «Интерфейс» как элемент языка ассоциируется с уменьями «определить интерфейс», «унаследовать интерфейс», «реализовать в классе метод интерфейса». Очевидно, что такие группы повторяют структуру синтаксиса языка программирования. Известны попытки представления этих структур в учебных целях в виде онтологий. С. Сосновский и Т. Гаврилова предложили онтологию для обучения языку С [16]. В [17] предложены принципы построения учебной онтологии для языка Java, а в [18] – для языков Java и C++. В этих онтологиях результаты преподавания (которые, в принципе, связаны с уменьями) ассоциируются с предметами (темами) преподавания и далее –с синтаксическими и семантическими структурами применяемого языка (языков) программирования.

В то же время встречаются примеры, демонстрирующие умения, вовлекающие несколько элементов языка, причём нередко в достаточно сложном сочетании. Примерами могут служить различные варианты синхронизации процессов, обработка особых случаев (exceptions), реорганизация хешированных коллекций, взаимодействие с базами данных. Такие примеры свойственны технической документации и ситуациям, встречающимся в практике опытных программистов и выходящим за рамки учебных планов.

Анализ известных разработок баз знаний по преподаванию языков программирования показывает:

- базы знаний этого класса основываются на онтологиях, обеспечивающих систематизацию элементов языков программирования и образовательного процесса; при этом онтологии для поиска и поддержки логического вывода не используются;
- онтологии строятся, главным образом, на отношении «часть-целое» между входящими в онтологию понятиями;
- онтологии, систематизирующие учебные примеры программ, не известны;
- визуализация онтологий осуществляется в форме «дерева» (графа, имеющего иерархическую структуру).

Опыт решения близких задач позволяет выделить в них следующие проблемные элементы:

- выбор классов понятий и отношений между понятиями, на которых строятся онтологии;
- определение номенклатуры онтологий, входящих в базу знаний;
- распределение понятий и отношений между онтологиями;
- выбор принципа визуализации онтологий.

[24] Энциклопедический словарь по психологии и педагогике [Электронный ресурс] // Академик, 2013. URL: https://psychology_pedagogy.academic.ru/10480 (дата обращения: 17.08.2022).

[25] Социально-педагогический словарь [Электронный ресурс] / М. Н. Бурмистрова, Л. Л. Васильева, Л. Ю. Петрова [и др.]. Саратов : Изд-во Саратовск. гос. ун-та, 2016. URL: http://elibrary.sgu.ru/uch_lit/505.pdf (дата обращения: 17.08.2022).

[26] Computational Pragmatics (CompPrag2016). Workshop at the 38th Annual Conference of the German Linguistics Society (DGfS) in Konstanz, February 24-26, 2016 [Электронный ресурс] // RUB Linguistik Startseite. Ruhr-Universität Bochum, 2022. URL: https://www.linguistics.rub.de/compprag2016 (дата обращения: 17.08.2022).

Далее в статье предлагается считать структуру тем преподавания вторичной, а рассматривать две связанных между собой сети: сеть умений и сеть примеров. Системообразующие отношения:

- умение реализуется примером (возможны множественные связи),
- уменье требуется для реализации примера,
- генерализация/конкретизация примеров,
- генерализация/конкретизация умений,
- пример является демонстрацией правила из спецификации языка.

На концептуальной основе этих сетей может быть построена оболочка базы знаний, предназначенной как для составления учебных пособий, так и для получения справок при составлении программ. База знаний выбирает последовательность умений и для каждого умения показывает один или несколько примеров, факультативно сопровождаемых релевантными выдержками из спецификаций по языку программирования.

База знаний должна позволять пользователю изменять состав умений и примеров. В простейшем варианте база знаний может быть построена на очевидных умениях и естественных примерах, исполнимых как самостоятельное приложение. В такой системе неизбежна повторяемость текстов в примерах. Нетривиальным развитием системы является декомпозиция примеров и умений. Принципиальной проблемой в этом случае оказывается глубина декомпозиции.

Рассмотренные сети могут быт построены на реляционных таблицах, но поскольку нетрудно предвидеть их высокую изменчивость в ходе эксплуатации, онтологии представляются более подходящим решением. В любом случае пример – это строка языка программирования с шаблонами или макросами или языка проектирования (UML). Она может храниться отдельно от онтологии.

Экспериментальная реализация базы знаний

В целях апробации предложенных принципов в магистерской диссертации А. А. Саликовой[27] разработан комплекс, включающий в себя следующие компоненты:

- оболочка базы знаний– модуль ProtoLego.exe;
- онтология умений – модуль Skills.xml;
- онтология определений языка программирования Java – модуль Defs.xml;
- библиотека модулей SkillsHeap –программ-примеров на языке Java, демонстрирующих умения;
- библиотека модулей DefsHeap – текстов определений элементов языка Java, применяемых в примерах.

Оболочка комплекса реализована на языке C# для исполнения в операционной среде Windows 10. Для решения поставленных задач приняты следующие подходы:

- онтология представляется экземпляром класса TreeView, который показывается в левом окне (онтология умений) или в правом окне (онтология элементов языка);
- онтологии хранятся в виде xml-файлов. Внешний интерфейс оболочки состоит из главного окна компьютера, клавиатуры и манипулятора мыши.

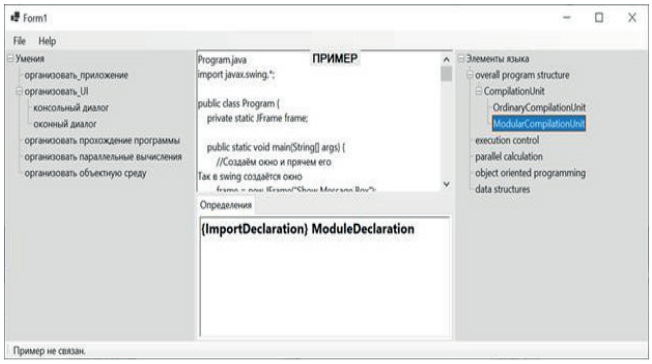
В главном окне находятся 4 вспомогательных окна:

- окно умений (левая часть главного окна),
- окно элементов языка (правая часть главного окна),
- окно примеров (верхнее окно в центре),
- окно определений элементов языка(нижнее окно в центре).

Границы между окнами подвижны, их можно сдвигать мышью. На Рис. 4 показан вид главного окна.

Узел онтологии в окне представляется словом или словосочетанием и сохраняется в свойстве Text узла дерева TreeView. Имя (Name) узла совпадает с этим словосочетанием, за исключением того, что пробелы заменены на подчёркивание (имя узла не может содержать пробелы). При создании нового узла по умолчанию предлагается словосочетание «Новый узел». Но сразу после создания узел находится в редактируемом (Editable) состоянии, и пользователь может с клавиатуры ввести, что считает нужным.

Результат двойного воздействия левой кнопкой мыши на узел онтологии зависит от того, присвоен ли узлу текст: если текст присвоен, то он выводится в окно как пример или определение, в зависимости от того, в какую онтологию входит узел; если текст на присвоен, то открывает окно диалога и пользователь может выбрать файл, содержащий текст, который станет присвоен узлу. Если пользователь отменит диалог, то узел останется без текста. Имя файла, содержащего присвоенный текст, хранится в экземпляре класса NodeBag. Этот экземпляр создаётся как раз тогда, когда узлу присваивается текст.



Р и с. 4. Главное окно оболочки базы знаний

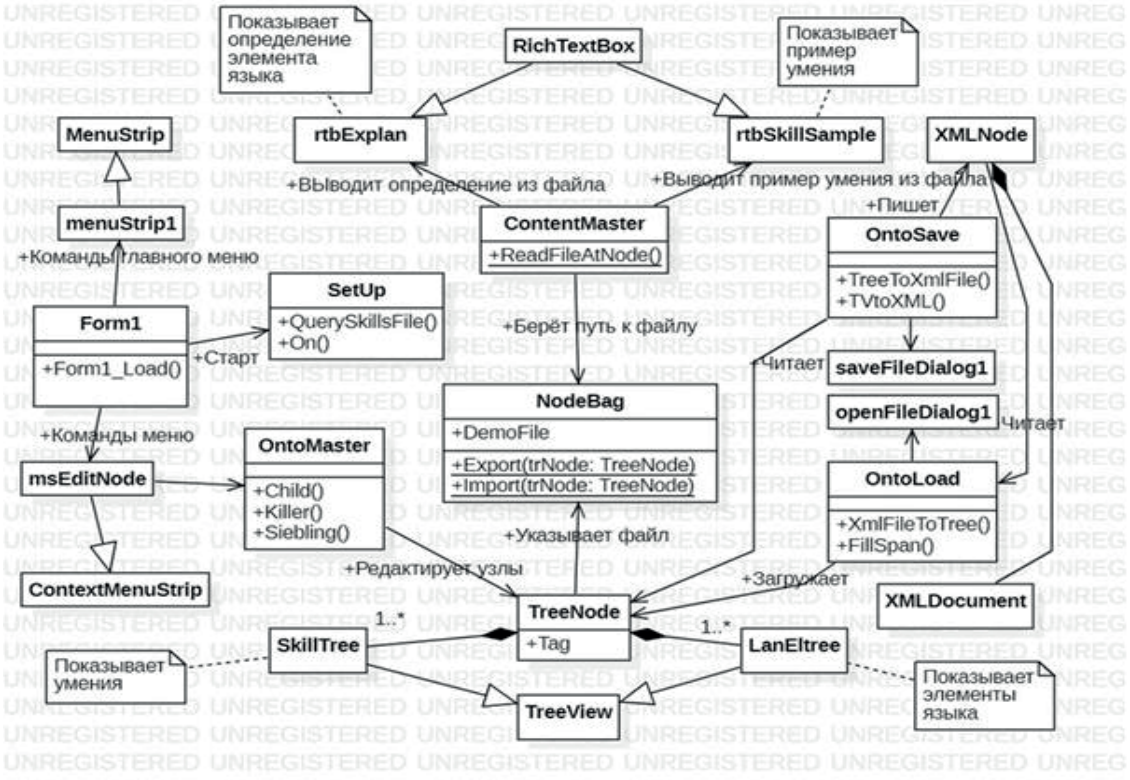
F i g. 4. Knowledge base shell main window

Полная структура классов оболочки показана на Рис. 5.

Дерево умений представлено классом SkillTree, а дерево элементов языка – классом LanTree, подклассами класса TreeView. Окно для показа дерева умений представлено экземпляром класса rtbSkillSample. Окно для показа дерева элементов языка представлено экземпляром класса rtbExplan. rtbSkillSample и rtbExplan являются подклассами класса RichTextBox.

ContentMaster обеспечивает ввод примеров и элементов языка в соответствующие окна из файловой системы. Классы OntoSave и OntoLoad, соответственно, сохраняют онтологии в виде xml-файлов и загружают их в окна.

[27] Саликова А.А. Разработка демонстрационно-образовательного комплекса по программированию на языке Java : маг. дисс. М. : МАИ, 2022. С. 79.



Р и с. 5. Диаграмма классов оболочки базы знаний

F i g. 5. Knowledge base shell class diagram

```
public static void TVtoXML(XmlNode xmlel, TreeNode trNode, XmlDocument
vdoc)
    {//При входе в метод гаранируется, что trnode не пуст.
      //        TreeNodeCollection tnColl = tnCol[tnCol.Count-1].Nodes;
      XmlNode nxXmlNode;
      do
      {//Обход в глубину
       nxXmlNode = vdoc.CreateElement(trNode.Name);
       xmlel.AppendChild(nxXmlNode);
       if (trNode.Tag != null)
          (trNode.Tag as NodeBag).Export(nxXmlNode, vdoc, trNode);
       if (trNode.Nodes.Count != 0) TVtoXML(nxXmlNode, trNode.Nodes[0], vdoc);
          trNode = trNode.NextNode;
      }
       while (trNode != null);
 }
```

Р и с. 6. Алгоритм преобразования онтологии из формата TreeView в формат XmlDocaument

F i g. 6. Algorithm for converting an ontology from the TreeView format to the XmlDocaument format

```
public static void FillSpan(Form1 form, TreeNodeCollection tnCol, XmlNode
xmlel)
      {
        //        TreeNodeCollection tnColl = tnCol[tnCol.Count-1].Nodes;
        TreeNode nxtreeNode;
        int bgindex;
        do
        {
          nxtreeNode = new TreeNode(xmlel.Name);
nxtreeNode.ContextMenuStrip = form.msEditNode;
          bgindex = tnCol.Add(nxtreeNode);
if (xmlel.HasChildNodes)
          {
            if (xmlel.ChildNodes[0].Name == "BAG")
            {
              NodeBag.Import(xmlel.ChildNodes[0], nxtreeNode);
              if (xmlel.ChildNodes.Count > 1)
                FillSpan(form, nxtreeNode.Nodes, xmlel.ChildNodes[1]);
            }
            else FillSpan(form, nxtreeNode.Nodes, xmlel.ChildNodes[0]);
          }
          xmlel = xmlel.NextSibling;
        }
        while (xmlel != null);
      }
```

Р и с. 7. Алгоритм преобразования онтологии из формата XmlDocaument в формат TreeView

F i g. 7. Algorithm for converting ontology from XmlDocaument format to TreeView format

Инициирование главного окна, представленного классом Form1, производится с участием класса SetUp, запрашивающего у оператора, с какой онтологии умений начнётся работа базы знаний. Класс OntoMaster обеспечивает редактирование онтологий, представленных деревьями в окнах.

Подклассы классов MenuStrip и Context MenuStrip отвечают за управление базой знаний посредством главного и контекстных меню. Интенсивное использование меню позволяет освободить пространство экрана от других органов управления в интересах более полного показа контента базы знаний.

При сохранении узла онтологии в xml-файл контент объекта NodeBag записывается в качестве содержимого тега BAG – первого и , может быть, единственного XML-подузла сохраняемого узла. Подузлы с этими тегами являются единственным нарушением структурной идентичности между деревом TreeView онтологии и XML-документа, в виде которого она сохраняется. При сохранении и загрузке онтологии древовидные структуры TreeView и XML-документа обходятся в глубину в модулях FillSpan и TVtoXML, представленных на Рис. 6 и 7, соответственно.

При создании узла онтологии свойству ContextMenuStrip присваивается ссылка на объект msEditNode для обеспечения вызова всплывающего (контекстного) меню при воздействии правой кнопкой мыши на узел.

Заключение

Примеры практического применения теоретических знаний важны в любом учебном процессе, но в обучении программированию они, можно утверждать без преувеличения, – не заменимы. Способность к эффективному синтезу текстов на языках программирования основывается на опыте разбора своих собственных и чужих программ. В статье предложен и опробован подход к собиранию наилучших образцов или чем-то примечательных программных текстов, называемых здесь «примерами программ», в базе знаний, построенной на основе онтологии программистских умений в сочетании с онтологией языков программирования. Оболочка базы знаний позволяет использовать онтологии различных языков и умений, редактировать и пополнять их, осуществляя, в целом, синтез синтаксических, семантических и прагматических знаний по языкам программирования и корпусам текстов программ.

References

[1] Derus S.R.M., Ali A.Z.M. Difficulties in learning programming: Views of students. In: 1st International Conference on Current Issues in Education (ICCIE'2012). Yogyakarta, Indonesia; 2012. p. 74-78. doi: https://doi.org/10.13140/2.1.1055.7441

[2] Lepp M., Kaimre J. Providing Additional Support in an Introductory Programming Course. In: 2022 IEEE Global Engineering Education Conference (EDUCON). Tunis, Tunisia: IEEE Computer Society; 2022. p. 210-216. doi: https://doi.org/10.1109/EDUCON52537.2022.9766661

[3] Usman M., Britto R., Börstler J., Mendes E. Taxonomies in software engineering: A Systematic mapping study and a revised taxonomy development method. *Information and Software Technology*. 2017;85:43-59. doi: https://doi.org/10.1016/j.infsof.2017.01.006

[4] Gahlyan P., Singh S.N. Analysis of Catalogue of GoF Software Design Patterns. In: 2018 8th International Conference on Cloud Computing, Data Science & Engineering (Confluence). Noida, India: IEEE Computer Society; 2018. p. 814-818. doi: https://doi.org/10.1109/CONFLUENCE.2018.8442878

[5] Tennent R.D. The denotational semantics of programming languages. *Communications of the ACM*. 1976;19(8):437-453. doi: https://doi.org/10.1145/360303.360308

[6] Tenenberg J., Fincher S. Students Designing Software: a Multi-National, Multi-Institutional Study. *Informatics in Education*. 2005;4(1):143-162. doi: https://doi.org/10.15388/infedu.2005.09.

[7] Tan P.-H., Ting C.-Y., Ling S.-W. Learning Difficulties in Programming Courses: Undergraduates' Perspective and Perception. In: 2009 International Conference on Computer Technology and Development. Kota Kinabalu, Malaysia: IEEE Computer Society; 2009. p. 42-46. doi: https://doi.org/10.1109/ICCTD.2009.188

[8] Turner R. Programming Language Specification. In: Computable Models. Springer, London; 2009. p. 1-7. doi: https://doi.org/10.1007/978-1-84882-052-4_22

[9] Turner R. Understanding Programming Languages. *Minds & Machines*. 2007;17(2):203-216. doi: https://doi.org/10.1007/s11023-007-9062-6

[10] Plotkin G.D. The Origins of structural operational semantics. *The Journal of Logic and Algebraic Programming*. 2004;60-61:3-15. doi: https://doi.org/10.1016/j.jlap.2004.03.009

[11] Astarte T.K., Jones C.B. Formal Semantics of ALGOL 60: Four Descriptions in their Historical Context. In: De Mol L., Primiero G. (eds.). Reflections on Programming Systems. Philosophical Studies Series. Vol. 133. Cham: Springer; 2018. p. 83-152. doi: https://doi.org/10.1007/978-3-319-97226-8_4

[12] Ellison C., Rosu G. An executable formal semantics of C with applications. *ACM SIGPLAN Notices*. 2012;47(1):533-544. doi: https://doi.org/10.1145/2103621.2103719

[13] Borgo S., Ferrario R., Gangemi A., Guarino N., Masolo C., Porello D., Sanfilippo E.M., Vieu L., Borgo S., Galton A., Kutz O. DOLCE: A Descriptive Ontology for Linguistic and Cognitive Engineering. *Applied Ontology*. 2022;17(1):45-69. doi: https://doi.org/10.3233/AO-210259

[14] Fellbaum C. WordNet(s). In: Encyclopedia of Language & Linguistics. Second Edition. Oxford: Elsevier; 2006. p. 665-670. doi: https://doi.org/10.1016/B0-08-044854-2/00946-9

[15] Miller G.A. WordNet: A Lexical Database for English. In: Proceedings of the workshop on Human Language Technology (HLT'93). Association for Computational Linguistics, USA; 1993. 409 p. doi: https://doi.org/10.3115/1075671.1075788
[16] Sosnovsky S., Gavrilova T. Development of Educational Ontology for C-Programming. *Information Theories & Applications*. 2006;13(4):303-308. Available at: http://www.foibg.com/ijita/vol13/ijita13-4-p01.pdf (accessed 17.08.2022).
[17] Ganapathi G., Lourdusamy R., Rajaram V. Towards Ontology Development for Teaching Programming Language. In: Proceedings of the World Congress on Engineering 2011 (WCE 2011). Vol. III. London, U.K.; 2011. Available at: https://www.iaeng.org/publication/WCE2011/WCE2011_pp1845-1848.pdf (accessed 17.08.2022).
[18] Pierrakeas C., Solomou G., Kameas A. An Ontology-Based Approach in Learning Programming Languages. In: 2012 16th Panhellenic Conference on Informatics. Piraeus, Greece: IEEE Computer Society; 2012. p. 393-398. doi: https://doi.org/10.1109/PCi.2012.78
[19] Gorodnyaya L.V., Andreyeva T.A. Programming paradigms in higher education. *Bulletin of the Novosibirsk Computing Center. Series: Computer Science*. 2015;(38):67-90. doi: https://doi.org/10.31144/bncc.cs.2542-1972.2015.n38.p67-90
[20] Saygin A.P., Cicekli I. Pragmatics in human-computer conversations. *Journal of Pragmatics*. 2002;34(3):227-258. doi: https://doi.org/10.1016/S0378-2166(02)80001-7
[21] Gagarin A.P., Pyzh'yanov I.L. Computer System Architecture for Teaching the Software Engineering in a Classroom Environment. *Pedagogical Informatics*. 2022;(3):202-216. Available at: https://www.elibrary.ru/item.asp?id=49564664 (accessed 17.08.2022). (In Russ., abstract in Eng.)
[22] Jurafsky D. Pragmatics and Computational Linguistics. In: Horn L.R., Ward G. (eds.) The Handbook of Pragmatics. Blackwell Publishing Ltd; 2006. p. 578-604. doi: https://doi.org/10.1002/9780470756959.ch26
[23] Bunt H. Computational Pragmatics. In: Y. Huang (ed.). The Oxford Handbook of Pragmatics. Oxford: Oxford University Press; 2017. p. 326-345. doi: https://doi.org/10.1093/oxfordhb/9780199697960.013.18
[24] Li X., Ma Z. Computational Pragmatics: A Survey in China and the World. In: Proceedings of the 2nd International Conference on Natural Language Processing and Information Retrieval (NLPIR 2018). New York, NY, USA: Association for Computing Machinery; 2018. p. 65-69. doi: https://doi.org/10.1145/3278293.3278304
[25] Pu Y., Ellis K., Kryven M., Tenenbaum J.B., Solar-Lezama A. Program Synthesis with Pragmatic Communication. In: Proceedings of the 34th International Conference on Neural Information Processing Systems (NIPS'20). Red Hook, NY, USA: Curran Associates Inc.; 2020. Article number: 1111. p. 13249-13259. Available at: https://dl.acm.org/doi/pdf/10.5555/3495724.3496835 (accessed 17.08.2022).
[26] Reuben C.-G., Goodman N., Potts C. An Incremental Iterated Response Model of Pragmatics. *Proceedings of the Society for Computation in Linguistics*. 2019;2:10. doi: https://doi.org/10.7275/cprc-8x17

Поступила 17.08.2022; одобрена после рецензирования 22.09.2022; принята к публикации 05.10.2022.
Submitted 17.08.2022; approved after reviewing 22.09.2022; accepted for publication 05.10.2022.

Об авторе:

Гагарин Андрей Петрович, профессор кафедры вычислительных машин, систем и сетей Института № 3 «Системы управления, информатика и электроэнергетика», ФГБОУ ВО «Московский авиационный институт (национальный исследовательский университет)» (125993, Российская Федерация, г. Москва, Волоколамское шоссе, д. 4), кандидат технических наук, профессор, **ORCID: https://orcid.org/0000-0002-0929-2834**, gagarin_ay@outlook.com

Автор прочитал и одобрил окончательный вариант рукописи.

About the author:

Andrey P. Gagarin, Professor of the Chair of Computers, Systems and Networks, Institute of Control Systems and Computer Science in Engineering, Moscow Aviation Institute (National Research University) (4 Volokolamskoe shosse, Moscow 125993, Russian Federation), Cand.Sci. (Eng.), Professor, **ORCID: https://orcid.org/0000-0002-0929-2834**, gagarin_ay@outlook.com

The author has read and approved the final manuscript.