УДК 004.43 БК 22.18 Г70

СИСТЕМАТИЗАЦИИ ПАРАДИГМ ПРОГРАММИРОВАНИЯ ПО ПРИОРИТЕТАМ ПРИНЯТИЯ РЕШЕНИЙ

Л. В. Городняя

Институт систем информатики им. А.П. Ершова Сибирского отделения Российской академии наук, Новосибирский государственный университет, г. Новосибирск

lidvas@gmail.com

Аннотация

Цель статьи – описание методики сравнения парадигм и языков программирования, отражающей выразительную силу языков, трудоёмкость реализации систем программирования и приспособленность к обоснованию практичных, объективных критериев декомпозиции программ, что можно рассматривать как подход к решению проблемы факторизации весьма усложнённых определений языков программирования и систем их поддержки. Представлены результаты анализа наиболее известных основных парадигм программирования и намечен подход к навигации в современном расширяющемся пространстве языков программирования. Систематизация парадигм учитывает особенности постановок задач программирования и семантические характеристики языков и систем программирования с акцентом на критерии качества программ и приоритеты в принятии решений при их реализации и обучении программистов.

Ключевые слова*: определение языков программирования, парадигмы программирования, классификация сложных определений, семантические системы*

ВВЕДЕНИЕ

Понятие «парадигмы программирования» (ПП) не имеет строгого определения, поэтому возникает вопрос о принадлежности новых подходов в программировании и ИТ к множеству ПП и об упорядочении такого множества. Парадигма программирования проявляется как образ мышления, связанный с компромиссом между особенностями решаемых задач, методами их решения в

форме программ, принятыми в ПП критериями качества программ и приоритетами принятия решений в процессе программирования. Такая особенность ПП позволяет понимать выбор парадигмы как процесс принятия, представления и отладки решений при постановке разных задач, поэтому естественно систематизацию ПП выполнить по сопоставлению с приоритетами и варьированием схем постановки задач и методов их решения. Наиболее ясная систематизация ПП в настоящее время позволяет выделять основные и производные ПП, дополненные комбинированными, вспомогательными и системообразующими или перспективно-стратегическими. Следует отметить, что академик Андрей Петрович Ершов основное внимание уделял именно стратегическим ПП, включающим фундаментальные, образовательные и технологичные. Множество основных ПП можно разделить на базовые, инструментально расширяющие и неограниченные в зависимости от наполнения семантических систем организации вычислений, работы с памятью, управления вычислениями и конструирования сложных данных.

Описания современных языков программирования (ЯП) обычно содержат список из 5–10 языков-предшественников и ряд ПП, поддержанных языком [1, 2]. Такая характеристика как правило не показывает, какие свойства ПП и черты предшественника фактически унаследованы определяемым ЯП и какие особенности являются действительно новыми. Для реальной оценки уровня новизны и практичности ЯП может быть полезной коллекция постановок задач с примерами их решения на разных языках в рамках различных парадигм. Определённое количество примеров обычно присутствует в описаниях ЯП и учебных пособиях. Чаще всего это программа печати приветствия, что позволяет быстро начать эксперименты с системой программирования (СП), но, учитывая, что большинство современных ЯП поддерживает механизмы ввода-вывода на уровне библиотечных вызовов, такие примеры слабо характеризуют язык.

В данной статье рассматривается систематизация парадигмальных особенностей определения ЯП на уровне семантических систем [3], классифицированных по постановкам задач и языковым средствам, используемым при их решении. Ещё в давние времена Николас Вирт отмечал важность соответствия поста-

новки задачи и используемого для её решения инструмента, особенно если удаётся уловить подобие обрабатываемых структур данных и алгоритмов их обработки, что теперь называют гомоиконностью. На основе такого соответствия можно выстроить пространство конструкций, поддержанных в определениях языков и систем программирования (ЯСП) и сопоставленных со сложностью постановок успешно решаемых задач. Полученное пространство может быть исходной структурой при выборе критериев декомпозиции программ с учётом особенностей развития постановок задач в процессе программирования их решений [4], расширения семантических систем ЯП и их уточнения при реализации СП [5].

Методика показана на материале четырёх классических базовых парадигм программирования в рамках ЯП высокого уровня без экскурса в языки низкого и сверх высокого уровней и распространена на более широкое пространство основных парадигм, особенно новых, ещё не получивших поддержки в достаточно известных языках программирования и признания в виде примеров отлаженных программных продуктов. Выполнен поверхностный анализ языков организации параллельных вычислений (ПВ), заслуживающих не менее двух парадигм, и DSL-языков, вероятно знаменующих переход прикладного программирования на качественно новый уровень.

Автор выражает благодарность участникам неформальной дискуссии о понятии «парадигма программирования», организованной Ан.В. Климовым 26-го сентября в рамках XXI Всероссийской конференции «Научный сервис в сети Интернет**».**

РЕЗУЛЬТАТЫ ПАРАДИГМАЛЬНОГО АНАЛИЗА

Анализ и сравнение большого числа ЯП разного уровня позволили выделить наиболее существенные характеристики для выражения парадигмальной специфики широкого класса новых ЯП (см. Таблицу 1)[1].

Интересно отметить, что парадигмальная характеристика многих долгоживущих ЯП исторически претерпела заметные изменения (см. Таблицу 2). Можно

[1] Перечни в Таблицах 1 и 2 опираются на открытые источники типа Википедий и учебных пособий, имеют предварительный характер. Перечни могут быть пополнены и уточнены. специалистами при более строгом изложении.

сказать, что отдельные парадигмы проявились постепенно, по мере накопления опыта решения определённых классов задач.

Таблица 1. Языки программирования XXI века (все мультипарадигмальные)

Год	*ЯП*	*Предшественники*[2]	*Поддержанные парадигмы*
2018	Dart	Java, JavaScript, CoffeeScript, Go	объектно-ориентированный каркас веб-приложений сценарный язык императивный рефлективный функциональный
2016	Kotlin	Java, Scala, Groovy, Gosu, C#, Python, ML	объектно-ориентированный язык JVM
2014	Swift	Objective-C, C++, Java, Rust, Scala, D, Python, Ruby, LLVM, Smalltalk, Groovy,	протоколо-ориентированный объектно-ориентированный функциональный императивный
2012	Rust	Alev, C++, Camlp4, C#, Common Lisp, NIL, Erlang, Haskell, Ruby, Hermes, Scheme, Napier, Napier88, Newsqueak, Sather, OCaml, Cyclone, Standard ML, Swift	параллельный функциональный императивный структурный системный процедурный свободное программное обеспечение
2009	Go	C, Limbo, Modula, Oberon, Pascal	императивный, на уровне значений, процедурный, скалярный, структурный
2007	Clozure	Lisp, ML, Haskell, Erlang, Prolog, Scheme, Java, Ruby	функциональный гомоиконный
2005	F#	OCaml, C#, Haskell	функциональный объектно-ориентированный обобщённый, императивный
2003	Scala	Java, Haskell, Erlang, Lisp, Standard ML, OCaml, Smalltalk, Scheme, Algol68	функциональный объектно-ориентированный императивный
2001	D	Си, C++, C#, Python, Ruby, Java, Eiffel	императивный объектно-ориентированный функциональный контрактный обобщённый, процедурный
2000	C#	C++, Java, Delphi, Модула-3, Smalltalk	объектно-ориентированный обобщённый процедурный функциональный событийный рефлективный

[2] Сохранена лексика источников.

Таблица 2. ЯП – основатели базовых парадигм программирования

Год	*ЯП*	*Поддержанные парадигмы*	*Сфера влияния*
1954 1958	Fortran, Algol-60 [3]	императивный параллельный процедурный модульный структурный процедурный обобщённый объектно-ориентированный	**ИПП** – **императивно-процедурное** ALGOL 58, BASIC, C, Chapel, CMS-2, Fortress, PL/I, PACT I, MUMPS, IDL, Ratfor
1958	Lisp	экспериментальный функциональный объектно-ориентированный процедурный рефлективный метапрограммирование	**ФП** – **функциональное** Clips, Common lisp, CLOS, Clu, Dylan, Forth, Scheme, Erlang, Haskell, Logo, Lua, Perl, POP-2, Python, Ruby, Cmucl, Scala, ML, Swift, Smalltalk, Factor, Clojure, Emacs Lisp, Eulisp, ISLISP, Wolfram Language
1960	APL	векторный функциональный структурный модульный	**ПВ** – **параллельные вычисления** A, A+, FP, J, K, LyaPAS, Nial, S, MATLAB, PPL, Wolfram Language
1962	Simula 67	объектно-ориентированный	**ООП (1980)** – **объектно-ориентированное**
1968	Forth[4]	императивный стек-ориентированный	Factor, RPL, REBOL, PostScript, Factor и другие конкатенативные языки 5
1968	Algol-68[6]	параллельный императивный	C, C++, Bome shell, KornShell, Bash, Steelman, Ada, Python, Seed7, Mary, S3
1972	Prolog	декларативный логический	**ЛП** – **логическое** Visual Prolog, Mercury, Oz, Erlang, Strand, KL0, KL1, Datalog
1970	Pascal	императивный структурный	**Структурное** Ada, Component Pascal, Modula-2, Java, Go, Oberon, Object Pascal, Oxygene, Seed7, VHD, Structured text

Мультипарадигмальность большинства долгоживущих и новых ЯП показывает необходимость более точной детализации зависимостей между старыми и новыми ЯСП. Наиболее объективные понятия программирования связаны с архитектурными моделями, с методами реализации СП и с классификацией реша-

[3] Algol-60 – именно с этого языка в нашей стране началось знакомство с языками высокого уровня, доминирование его оттеснилось появлением реализаций языка Fortran.

[4] Forth – типовой механизм реализации работы с выражениями в разных ЯП.

[5] Языки, в которых программы строятся как конкатенации функций.

[6] Algol-68 представляет результат хорошо продуманной унификации и ортогонализации основных понятий программирования.

емых задач. Для показа особенностей ПП удобно выделять концептуальные монопарадигмальные ЯП или подъязыки и приводить критерии успешного применения ПП с оценкой результатов на примерах программ, подтверждающих признание практикой программирования [5].

Таблица 3. Наиболее востребованные ЯП (все мультипарадигмальные)

Год	*ЯП*	*Предшественники*	*Поддержанные парадигмы*	*Сфера влияния*
1972	C	B (BCPL, CPL), Algol-68, Assembly, PL/I, FORTRAN	императивный на уровне значений процедурный скалярный структурный	Numerous, AMPL, AWK, csh, C++, C--, C#, Objective-C, D, Go, Java, JavaScript, Julia, Limbo, LPC, Perl, PHP, Pike, Processing, Python, Ring, Rust, Seed7, Vala, Verilog (HDL), Nim, Cyclone, BitC
1974	SQL	Datalog	декларативный строгий	Agena, CQL, LINQ, Windows, PowerShell, ABAP
1983	Objective-C	Smalltalk, C	объектно-ориентированный рефлексивно-ориентированный	Java, Objective-J, Swift
1983	C++	C, Simula, Algol 68, CLU, ML, Ada	объектно-ориентированный процедурный метапрограммирование процедурный функциональный обобщённый язык свободной формы	C#, D, Falcon, Java, Lua, Perl, Pike, Python, Rust, Vala
1987	Erlang	ML, Miranda, Ada, Modula-2, CHILL, Prolog	параллельный функциональный декларативный открытое программное обеспечение свободное программное обеспечение	Rust, Sparkel
1991	Python	ABC, Modula-3, Lisp, Tcl, Smalltalk, C, Java, Icon	объектно-ориентированный рефлективный императивный функциональный аспектно-ориентированный динамический	Ruby, Boo, Groovy, ECMAScript, CoffeeScript, Swift, Nim

Продолжение Таблицы 3. Наиболее востребованные ЯП (все мультипарадигмальные)

Год	*ЯП*	*Предшественники*	*Поддержанные парадигмы*	*Сфера влияния*
1991	Html	SGML	Язык разметки и переписывания	shtml
1993	R	Common Lisp, S, Scheme, XLispStat	векторный императивный аспектно-ориентированный процедурный функциональный рефлективный	Julia
1995	Java	C++, C, C# Ada, Simula 67, Mesa, Smalltalk, Objective-C, Object Pascal, UCSD Pascal, Oberon, Eiffel, Modula-3, Simula	императивный на уровне значений объектно-ориентированный процедурный скалярный	C#, Ceylon, D, E, ECMAS, cript, Groovy, Process, ing, Scala, Vala, X10
995	JavaScript	Lua, Self, Си, Scheme, Perl, Python, Java, AWK, HyperTalk	объектно-ориентированный (прототипный) обобщённый функциональный императивный аспектно-ориентированный событийно-ориентированный	Objective-J, Dart, TypeScript
1995	Ruby	Ada, Dylan, Perl, Python, Smalltalk, C++, Клу, Eiffel, Lisp, Basic, Lua	функциональный императивный рефлективный	Falcon, Groovy, Objeck
1995	PHP	Perl, C, C++, Java, Tcl	сценарный, процедурный объектно-ориентированный императивный интерпретируемый свободное программное обеспечение	Falcon

Из огромного множества можно выделить небольшое число ЯП, привлекающих внимание интересными сочетаниями изобразительных средств и особенностями семантики, влияющими на развитие основных ПП.

Таблица 4. Интересные ЯП (все мультипарадигмальные)

Год	*ЯП*	*Предшественники*	*Поддержанные парадигмы*	*Сфера влияния*
1969	Setl	Algol-60	императивный процедурный, структурный, объектно-ориентированный	SETL2, ISETL, SETLX, ABC
1983	Sisal	VAL, Pascal, C, Fortran	функциональный, потоков данных	Haskell, SAC
1987	Perl	Си, AWK, Shell, Sed, Lisp	императивный объектно-ориентированный функциональный	Ruby, PHP, Groovy
1990	Haskell	ML, Standard ML, Lazy ML, Miranda, Lisp, Scheme, ISWIM, FP, АПЛ, Hope, Hope+, SISAL, Orwell*, Id	функциональный, ленивый, модульный	Agda, Bluespec, Clojure, C#, Cat, Cayenne, Clean, Curry, Epigram, Escher, F#, Factor, Isabelle, Java Generics, Idris. LINQ, Mercury, Ωmega, Perl 6, Python, Qi, Scala, Swift, Timber, Visual Basic 9.0
1991	Visual basic	QuickBasic, BASIC	процедурный объектно-ориентированный компонентно-ориентированный событийно-ориентированный	Visual Basic .NET, REALbasic, Gambas, Xojo, Basic4ppc
1993	LUA	C++, CLU, Modula, Scheme, SNOBOL	императивный функциональный объектно-ориентированный (прототипный) скриптовый встраиваемый	GameMonkey, Io, JavaScript, Julia, MiniD, Red, Ring, Ruby, Squirrel, MoonScript, C--
1993	Brainfuck	FALSE	императивный на уровне значений эзотерический	BrainSub, Brainfork, Brainloller, COW, Ook, Pbrain, Smallfuck, Spoon, LOLCODE, Whitespace, DoubleFuck, Feckfeck
1995	Delphi	Object Pascal, C++	императивный структурированный объектно-ориентированный компонентно-ориентированный высокоуровневый	C#, Java
1996	Ocaml	Standart ML, Caml Light	функциональный объектно-ориентированный императивный	F#, JoCaml, MetaOCaml, OcamlP3l
2002	Falcon	*C++, Lisp, Lua, PHP, Perl, Python, Ruby, Smalltalk*	обмен сообщениями объектно-риентированный прототипный, процедурный функциональный, табличный	
2003	Groovy	Java, Ruby, Python, Perl, Smalltalk	объектно-ориентированный императивный, сценарный	
2012	TypeScript	JavaScript, C#	объектно-ориентированный обобщённый функциональный императивный, прототипный аспектно-ориентированный событийно-ориентированный	

СЕМАНТИЧЕСКИЕ СИСТЕМЫ ОСНОВНЫХ ПАРАДИГМ

Рассматривая любые семантические системы, важно отметить разницу в характере выполнения функций таких систем в различных контекстах и варианты в выборе методов их реализации. Так, для любого множества данных D, представляющих значения V произвольной природы, реализационно различимы схемы функций для методов вычислений E, средств доступа к памяти M через адреса N, особенностей управления вычислениями C и коммуникации или обратимой комплексации и структурирования данных S. Это приводит к представлению об основных категориях семантических систем различно реализуемых схем функций F. Исторически на уровне аппаратуры такие категории систем обладали кумулятивным эффектом в порядке «DEMCS» (см. Таблицу 5) – представление чисел, арифмометр, калькулятор, дифференциальный вычислитель, компьютер, причём, аппаратные подсистемы могут взаимодействовать каждая с каждой. Парадигмы программирования можно отличать по приоритетам выбора решений при определении категорий семантических систем в процессе программирования, отмечая парадигмальные различия общих понятий в каждой модели, зависящие от критериев качества программ.

Таблица 5. Ряд категорий семантических систем аппаратного уровня

Подсистема	*Примечание*
D: данные	Данные из множества D представляют значения из V и шкалу прерываний
E: вычисления	Операции по двум-одному значению производят одно или два значения
M: память	Соответствие между адресами из множества N и хранимыми по этим адресам представлениями из множества D для значений из V допускает разные методы доступа к элементам памяти, включая замену хранимых значений, за исключением адреса 0
C: управление	Сравнение значений с нулём позволяет управлять ходом вычислений наряду с передачами по меткам и обработкой прерываний, не считая перехода по порядку
S: коммуникации	Конструирование сложных данных учитывает возможности команд адресации в памяти

Такая разница может быть показана для классических базовых ПП (см. Таблицы 6–9). Для ИПП данные – это адреса и хранимые коды значений, в ООП появляются хранимые методы и сигнатуры объектов, в ФП вместо адресов в памяти может использоваться связывание с любым значением, а в ЛП – с идентификатором. В ИПП и ООП операции в основном унарные или бинарные, а в ФП и ЛП

имеется и произвольная арность. Истинностные значения в ЛП включают специальное значение «ESC», позволяющее отличать нормальные значения предикатов от неуспеха в вычислениях, а ФП может в качестве истины использовать любое значение отличное от пустого списка – «NIL». Структуры данных в ИПП могут не рассматриваться как значения, обрабатываемые базовыми средствами, а в ФП такие структуры обрабатываются без особых ограничений. Можно отметить, что базовые ПП различают представления значений, формул, сигналов и контекстов.

При подготовке императивно-процедурной программы доминирует критерий эффективности, понимаемый как оптимальное соотношение пространственно-временных характеристик программы. Поэтому важнейшими считаются средства работы с памятью, в которой размещаются данные и результаты их обработки (1: M). Управление процессом обработки представляется с помощью конструкций ветвления, использующих операции сравнения и предикаты над данными (2: C). Обработка данных рассматривается как изменение состояний памяти в порядке выполнения вычислений (3: E). При необходимости можно реорганизовывать структуру данных и программы (4: S) (см. Таблицу 6).

Таблица 6. Парадигмальная шкала ИПП **(MCES)**

Подсистема	*Приоритет*	*Примечание*
D: данные	0	Значения ограничены размерами их представлений в регистрах памяти по адресам из N. Шкала прерываний не представлена
E: вычисления	3	Операции различаются на унарные и бинарные с одним результатом
M: память	1	Работа с памятью без акцента на особенности нуля, разнообразие методов доступа к памяти и обработки прерываний
C: управление	2	Кроме аппаратного сравнения значений с нулём, при управлении ходом вычислений используются приоритеты операций и скобки в выражениях наряду с передачами по меткам, но без обработки прерываний
S: комплексация	4	Можно конструировать сложные данные и выбирать их элементы, используя возможности команд индексной адресации в памяти

В центре внимания ФП полнота семейства методов организации вычислений, доступных для эксперимента. Поэтому всё начинается с выбора базовых средств для обработки символьных представлений сущностей заданной пред-

метной области (1: E). Конструирование сложных объектов освобождено от обязательности соседства элементов (2: S). Работа с памятью в таком случае может не требовать привязки к физическим адресам, а ограничиться представлением ассоциирующей функции над парами данных любой природы (3: M). Управление процессом вычислений может рассматриваться как функция над фрагментами программы, рассматриваемыми как разновидность значений (4: C) (см. Таблицу 7).

Таблица 7. Парадигмальная шкала ФП **(ESMC)**

Подсистема	*Приоритет*	*Примечание*
D: данные	0	Представления значений не ограничены по размеру и сложности, включая функции
E: вычисления	1	Некоторые операции могут обрабатывать любое число параметров и вырабатывать ряд значений, при необходимости объединяемых в сложное данное
M: память	3	При обработке сложных данных старые значения не изменяются, а новые значения располагаются в памяти независимо, причём соответствие между ассоциированными данными хранится в памяти, допуская изменение ассоциации
C: управление	4	Любое вычисление можно заблокировать или запустить. Программа может содержать точки ветвления из произвольного числа ветвей, выбираемых сравнением результата с «нулём» (NIL), входящим в множество значений
S: структуры	2	Можно конструировать сложные, равноправные с элементарными, данные, все элементы которых доступны с помощью функций в любом выражении

В случае ЛП важно показать существование хотя бы одного полезного решения задачи. Это приводит к доминированию логики недетерминированного поиска выполнимых решений (1: C). Формально вычисляются варианты возможных решений (2: E) в произвольном порядке. Но в качестве структур используются образцы (3: S), позволяющие управлять выбором вариантов. Именуются фрагменты с фиксированным числом параметров (4: M), от необходимости имён которых техника образцов освобождает (см. Таблицу 8).

Ведущий критерий качества программ в ООП – сопоставимость иерархии классов объектов с иерархией понятий в области приложения программ, позволяющая готовые программные решения пополнить новыми для расширения прежней области приложения. Поэтому здесь всё начинается с определения

иерархии классов объектов (1: S), размещаемых по фиксированным адресам в памяти (2: M), применяемым как указатели ради достижения некоторого уровня эффективности. Управление процессом обработки данных использует сопоставление классов объектов и допустимых методов обработки объектов, размеченных правами доступа из разных частей программы (3: C). Вычисления происходят лишь при успешном сопоставлении и соответствии прав доступа к объектам (4: E) (см. Таблицу 9).

Таблица 8. Парадигмальная шкала ЛП **(CESM)**

Подсистема	*Приоритет*	*Примечание*
D: данные	0	Представления фактов или правил
E: вычисления	2	Попытки вычисления, дающего или результат, или сигнал о не выполнимости, что приводит к дальнейшему перебору вариантов
M: память	4	Некоторые правила могут иметь имена с указанием числа параметров, что позволяет их использовать как функции
C: управление	1	Недетерминированный перебор вариантов для выбора выполнимого варианта, дающего результат, отличный от «ESC»
S: шаблоны	3	Сопоставление сложных данных с образцом позволяет выделять элементы для выбора ветви вычислений

Таблица 9. Парадигмальная шкала ООП **(SMCE)**

Подсистема	*Приоритет*	*Примечание*
D: данные	0	Данные представляют не только значения, но и методы их обработки
E: вычисления	4	Операции бинарные и унарные, также как любые вычисления, можно перегрузить, добавив обработку возможных прерываний
M: память	2	Дозированный доступ к элементам объектов сопровождается механизмами неявных обработчиков ситуаций, адреса могут быть значениями
C: управление	3	Выполнимость методов над объектами обусловлена проверкой их совместимости и прав доступа по иерархии классов
S: структуры	1	Классы объектов приспособлены к доопределению и наследованию по иерархии классов

Подробный анализ семантики ООП, сопровождённый сопоставлением с другими ПП и частичной формализацией основных механизмов реализации ЯП, представлен в работе [6].

Показывая отличия в схемах определения функций для разных категорий семантических систем в зависимости от ПП, связанных с различием критериев

качества программ и приоритетов используемых в их реализации средств, следует отметить, что переход от ЯП к СП обычно сопровождается расширением числа поддержанных ПП. При определении языка Haskell это привело к понятию «монада», позволяющему любому ЯП достигать практичности на уровне СП, что обычно и выполнялось с помощью библиотечных модулей.

Кроме сравнительно ясных классических базовых ПП, есть основания выделять основные инструментальные системно **расширяющие** ПП, нацеленные на подготовку и конструирование программ, операционных систем (ОС) и баз данных (БД), поддержку работы с файлами и разнообразными конфигурациями устройств, а также обеспечение обратной связи при выполнении любых программ (см. Таблицы 10–13).

Все *расширяющие* ПП, некоторые из них ещё не получили свои имена, работают со много более сложными элементами, обладающими своей жизнью, допускающими включение во многие системы и конфигурации, в которых возможно изменение их состояния. Представления данных включают в себя, кроме сложных структур данных, формальных определений и кодов, процессы, устройства, роли участников и комплексы. Методы обработки элементов и их взаимодействия подчинены более жёстким требованиям правильности, что влечёт поддержку улучшения элементов по частям, то есть целенаправленного развития по мере выявления ошибок или необходимости в повышении эффективности. Имеет место разделение труда по уровню квалификации и ответственности.

Прежде всего это ПП, поддерживающие системное программирование, использующее синтаксически и грамматико ориентированную обработку определений ЯП при конструировании СП (VDM, BNF в инструментах типа Lex/YACC и их аналогах во многих новых ЯП), метапрограммирование (Рефал), языки и системы программирования и разработки программных инструментов (Bliss, ЯРМО, С), создание промежуточных форм для поддержки особо сложных работ (внутренний язык системы БЕТА), отладки программ и их тестирования.

В области разработки СП характерно деление на разработчиков СП и применяющих СП программистов, работающих в рамках базовых ПП и при необходимости подключающих расширяющие ПП в форме побочных эффектов библиотечных модулей. Появление большого количества DSL-языков на базе технологии

Clang-LLVM показывает новый уровень сформированности парадигм системного программирования, стирающий эту границу. Предметно-ориентированные DSL-языки заслуживают отдельного рассмотрения именно как новый уровень программистского языкотворчества. Если при развитии аппаратуры успешный опыт накапливается в форме системы команд, а в обычных ЯП накопление опыта программирования выполняется в форме отлаженных процедур, то конструирование DSL – механизм накопления опыта решения задач применения ИТ в форме языков программирования, что допускает включение такого механизма в инструментально расширяющие ПП.

Таблица 10. Парадигмальная шкала средств разработки СП **(ECMS)**

Подсистема	*Приоритет*	*Примечание*
D: данные	0	В качестве значений выступают определения реализуемого ЯП и система команд целевой машины, на которой предстоит выполнять программы, подготовленные на базе данного ЯП
E: вычисления	1	Синтаксически управляемый автомат с переводом строит переход от ЯП к его абстрактному представлению, более удобному для дальнейшей обработки программ и кодогенерации. Константные вычисления и символьные преобразования могут выполняться в этом процессе и заменяться на результат с целью повышения эффективности программ
M: память	3	Заполняется встроенная база данных, формируемая как таблицы, сопоставляющие адреса или идентификаторы отдельным конкретизациям понятий, выделяемых при анализе текста, или их атрибутам. Возможно привлечение промежуточных представлений для решения сложных технических задач. Строится схема программы, которая может быть достаточной для её выполнения или для кодогенерации
C: управление	2	Проверка текстов на принадлежность ЯП сопровождается диагностическими сообщениями и, возможно, рекомендациями по продолжению разбора текста. Результаты лексического, синтаксического и семантического анализа проверяются на соответствие шаблонам предстоящей кодогенерации. Определение ЯП работает подобно представлению типа данных. Кодогенерация может выполняться автономно. При выполнении созданной программы могут быть дополнительные проверки правильности вычислений.
S: структуры	4	Используются графовые и кодовые представления понятий, выделяемых при анализе текстов на принадлежность ЯП и выводе атрибутов, полезных для предстоящей кодогенерации и исполнения программы

Происходившее автономно развитие средств и методов работы с базами данных в настоящее время активно интегрируется в общий контекст ИТ, создаёт реализационную поддержку методам искусственного интеллекта и представле-

ния знаний с привлечением экспертов (GPS, RDF, OWL), является основой мощного пространства производственных и бизнес технологий, особенно на базе интернет-сервисов. Сформировано разделение труда на администраторов, ответственных за хранение данных, и клиентов, интересующихся их содержанием, способных его применять и уточнять.

Отдельные методы из этой сферы, изначально накапливающей решения по надёжности обработки данных, перемещаются в пространство решений при создании новых языков программирования, особенно поддерживающих параллелизм.

Таблица 11. Парадигмальная шкала СУБД **(MSEC)**

Подсистема	*Приоритет*	*Примечание*
D: данные	0	Данные представляют объекты реального мира и их характеристики, а также сведения о правах доступа к данным
E: вычисления	3	Операции обеспечивают формирование запросов, направленных на доступ к накопленным данным, включая их обработку
M: память	1	Хранение содержательных представлений в форме отношений дополняется использованием имён, индексов, функций, позволяющих повышать эффективность доступа к данным. Долговременным хранилищем данных являются файлы. На время обработки данных создаётся их временная копия в оперативной памяти.
C: управление	4	Кроме прав доступа к хранимым данным используются предикаты и логические выражения. Имеются рекомендации по учёту так называемых «нормальных форм», позволяющих повысить эффективность хранения данных и доступа к ним, причём некоторые из них дают одновременный выигрыш по скорости и по объёму памяти.
S: структуры	2	Отношения между хранимыми данными могут быть представлены как записи, организованные в таблицы, графы, иерархии (деревья), сети

Работы по операционным системам весьма осторожно переходят к использованию языков высокого уровня, преимущественно скриптовых и интерпретируемых, пригодных для реагирования в динамике реального мира. Возникает смежная линия компонентного программирования (Com/Dcom, Corba, SOAP, .Net), поддерживающая ООП при решении вопросов, требующих повышенной квалификации. Обычно разделяют обычный и привилегированный доступ (Таблица 12)

Таблица 12. Парадигмальная шкала ОС **(CMSE)**

Подсистема	*Приоритет*	*Примечание*
D: данные	0	Данные представляют задания, процессы, файлы, каталоги, устройства, протоколы и настроечные значения, а также права доступа.
E: вычисления	4	Работу операций выполняют доступные из командной строки команды уровня ядра, дополняемые представлениями программируемых заданий уровня оболочки
M: память	2	Основное хранение данных в файлах сопровождается использованием очередей, приоритетов, шкалы допустимых прерываний и настроечных значений, размещаемых при загрузке системы и допускающих уточнение привилегированным системным администратором
C: управление	1	Система функционирует как взаимодействие приаппаратного ядра и пользовательской оболочки. Имеется контроль прав доступа с учётом приоритетов и успеха-провала в ранее выполненных действиях, включая наличие-отсутствие необходимых устройств и файлов. Выполнение заданий инициируется с уровня командной строки.
S: структуры	3	Основная структура – иерархия файлов, дополненная очередями, строками, каналами и кодами, включая данные о регистрации пользователей и их правах доступа

Расширение спектра новой аппаратуры, пригодной для массового применения без технического сопровождения, привела к задачам формирования программно-аппаратных многопроцессорных комплексов для расширения сфер применения ИТ, как правило силами группы технической поддержки. (Таблица 13).

Не менее заметно выделяется группа **неограниченных** коммуникационно интерфейсных ПП, поддерживающих обработку большеобъёмных данных (bigdata, sematic-web, rdf), дистанционную работу в сетях, сервис-ориентированное программирование на базе языков разметки и переписывания (html, XML, PHP), параллельные, векторно-ориентированные для обработки массивов (APL) или поддерживающие теоретико-множественные инсерционное механизмы, включая динамические вставки-замены (SETL) и высокопроизводительные вычисления на суперкомпьютерах (OpenMP, mpC) и мобильных устройствах (см. Таблицы 14–17).

Языков программирования, поддерживающих неограниченные ПП, пока создано немного, но возможно их число будет возрастать, пока не получит удоб-

ной формы синхронизация процессов над общей памятью. За редким исключением потребность в неограниченных ПП реализуется в форме отдельных специализированных программных инструментов.

Таблица 13. Парадигмальная шкала многопроцессорных комплексов **(SECM)**

Подсистема	*Приоритет*	*Примечание*
D: данные	0	Данные представляют процессоры и программные модули, организованные в комплекс совместно эксплуатируемого оборудования, конфигурация которого может динамически изменяться
E: вычисления	2	Потоки действий и реагирование на сигналы от оборудования, программ и операторов, выработка сигналов и проявление реальных характеристик комплекса
M: память	4	Распределение оборудования и дозированный доступ к ресурсам и процессорам, обладающим ограниченным временем жизни, зависящим от включения-выключения и времени хранения сигналов
C: управление	3	Выполнимость действий обусловлена проверкой условий готовности устройств, наличием ресурсов, временем хранения данных в буферах памяти и пропускной способностью процессоров, допускающих параллельное использование
S: структуры	1	Конфигурации устройств, включая произвольное число процессоров и программ, и потоки действий, позволяющих ими оперировать

Переход к обработке большеобъёмных данных пока развивается на уровне специализированных языков и инструментов, практику применения которых несколько сдерживает отсутствие методов декомпозиции, преобразования и проектирования сложных слабо структурированных формирований, обладающих трудно контролируемой динамикой обновления. Большинство решений сводится к выбору визуализации графов (см. Таблицу 14).

Таблица 14. Парадигмальная шкала большеобъёмных данных – bigdata **(EMSC)**

Подсистема	*Приоритет*	*Примечание*
D: данные	0	Большие объёмы данных, возможно неструктурированных, представляют на многих машинах в сети, алгоритмы анализа таких данных могут иметь высокую временную сложность и содержательное разнообразие. Используются формы представления графов с большим числом вершин.
E: вычисления	1	Операции визуализации и анализа большеобъёмных данных дополнены средствами свёртки и редукции (MapReduce)для повышения наглядности и оптимизации, а также навигации с использованием поисковиков
M: память	2	Доступ к информационным ресурсам, хранимым на разных серверах, допускающих манипулирование большеобъёмными данными с некоторыми гарантиями условий хранения в стиле грид и облачных технологий
C: управление	4	Выполнимость отдельных методов манипулирования большеобъёмными данными и их визуализацией постепенно приводит к формированию практичных технологий и инструментов, технологических возможностей анализировать огромные массивы данных преимущественно средствами массово-параллельной обработки неопределённо структурированных данных и поддержки информационно-технологических решений
S: структуры	3	Графовые структуры и реальные конфигурации машин, видимых в сети с выделением серверов, регулирующих доступ к информационным ресурсам, или предоставляющим некоторые простаивающие мощности для общего употребления

Всеохватность методов дистанционного доступа резко влияет на качество жизни большинства слоёв населения, что сопряжено с вопросами социальной ответственности. Проявляются эффекты механизмов типа моды и рекламы. (см. Таблицу 15).

Постановки задач ПВ учитывают существование областей приложения, в которых недостаточна скорость получения результатов по доступным программам решения конкретных задач. Парадигмы этого направления находятся в стадии формирования из-за сложности перехода к масштабируемым решениям языков сверх высокого уровня, допускающим автоматическую настройку на реальные конфигурации оборудования, а также из-за отсутствия традиции представлять течение времени в математических моделях ЯП. Некоторые проблемы решает фрагментное программирование, предлагающее декомпозировать программу на схему управления и наполняющие её фрагменты. (см. Таблицу 16).

Таблица 15. Парадигмальная шкала дистанционного доступа **(MECS)**

Подсистема	*Приоритет*	*Примечание*
D: данные	0	Данные представляют заполненные в соответствии с разными форматами из тэгов, наполнения и адресов, информационные ресурсы и сведения об их владельцах и заинтересованных лицах
E: вычисления	2	Операции встраиваются и показываются в виде меню, состав которых может регулироваться специальными настройками
M: память	1	Доступ к элементам данных подчинен некоторой дисциплине, внешне маскируемой разными изобразительными средствами, включая цвет, звуки и форматы интерфейса. Реально используются неявные механизмы дублирования и восстановления данных, обеспечивающие стабильность функционирования
C: управление	3	Выполнимость методов обработки данных отчасти зависит от динамики функционирования серверов в сети, но в некоторых пределах возможно управление маршрутизацией
S: структуры	4	Сети, включающие в себя сервера, способные поддерживать маршруты доступа к данным в сети, адресуемые ресурсы, окна, страницы, линейки, шкалы-визуализаторы процессов

Таблица 16. Парадигмальная шкала параллельных вычислений **(CSME)**

Подсистема	*Приоритет*	*Примечание*
D: данные	0	Данные представляют действия, спусковые условия их выполнения и обрабатываемые значения и результаты, включая события, происходящие при выполнении действий и в окружающем контексте
E: вычисления	4	Операции, кроме обычных арифметических над значениями и доступа к памяти соответственно дисциплине, позволяют формировать сети и комплексы и извлекать из них составляющие, а также просачивать действия относительно структур
M: память	3	Память может быть неоднородной, элементы которой подчинены разным дисциплинам доступа и временным характеристикам реагирования
C: управление	1	Выполнимость действий регулируется спусковыми условиями, возможно зависящими от успеха предшествующих действий. Порядок одновременно выполнимых действий зависит от реализации, включая возможности синхросетей
S: структуры	2	Комплексы данных и сети действий, связанных со спусковыми условиями срабатывания и наполнением входных данных. Сеть может быть иерархической и структурированной, не исключая связи между уровнями и схемы синхронизации функционирования подсетей (синхросети)

Высокопроизводительные вычисления (ВПВ) расширяют спектр методов

управления вычислениями и критериев эффективности программ благодаря возможности использования процессорных резервов и наследования ранее отлаженных программ, как правило, посягая на их неприкосновенность не только методами распараллеливания, но и прямым редактированием. Это позволяет рассматривать ВПВ как бесспорный полигон для признания потенциала методов верификации, есть основания такие методы рассматривать как важную составляющую неограниченных парадигм.

В последние годы проявляются причины обуславливать конструирование средств верификации программ формализацией используемых ПП, а программистские проекты сопровождать обоснованием выбора не только инструментария, но и ПП, чтобы избегать межпарадигмальных конфликтов, чреватых трудно уловимыми ошибками, связанными с изменением обстановки функционирования программируемых компонентов. (см. Таблицу 17).

Таблица 17. Парадигмальная шкала суперкомпъютерных вычислений **(SCEM)**

Подсистема	*Приоритет*	*Примечание*
D: данные	0	Данные представляют готовые программы и измерительные модули, включая датчики энергопотребления, временные параметры срабатывания действий, события по срабатыванию разных действий
E: вычисления	3	Измерение производительности, выделение в программе критических участков, сравнение производительности, распараллеливание программ, приведение программ к распараллеливаемой форме, протоколирование событий и характеристик производительности или загрузки оборудования и программ, верификация схем и фрагментов программы
M: память	4	Неоднородная многоуровневая память, возможно с каналами связи с процессорами, дисциплиной доступа и коммуникацией по обновлению хранимых данных, включая их копирование и восстановление (транзакции и сборка мусора)
C: управление	2	Учёт результатов верификации и измерения производительности, эффективности и энергопотребления при распределении ресурсов и потоков действий, прогон программ по протоколу, реагирование на события согласно регламенту или спецификации,
S: структуры	1	Сети и схемы, приспособленные к включению в них действий и фрагментов программ

ПРОИЗВОДНЫЕ ПАРАДИГМЫ

Описание **производных** ПП можно сделать относительным и, следовательно, более **лаконичным**, выражая разницу с базовой ПП. Можно сказать, что производная ПП является проекцией базовой ПП на особенности постановок задач некоторой области приложения. Обычно в проекции видоизменяются наиболее важные элементы ПП. Вариации моделей семантических систем, поддерживающих производные парадигмы, можно использовать как объективные параметры при факторизации определений ЯСП и декомпозиции программ, начиная с учёта особенностей постановок задач.

Постановки задач базовой ИПП начинаются с определённого алгоритма решения актуальной задачи. Необходимо получить программу реализации алгоритма с практичными пространственно-временными характеристиками на конкретном доступном оборудовании. Производные ИПП выделяют разные методы представления данных в памяти (М) и организации порождаемых программой последовательных процессов, дополненных обработкой прерываний, поддержкой ПВ и сетевых процессов (С):

- автоматное – без процедур (блок-схемы);
- скалярное – без структур данных (Cobol);
- процедурное – библиотеки (Algol, PL/1);
- структурное – регулярная логика и типы данных (Pascal);
- сборочное – крупноблочность (assembler);
- модульное – повторное использование (Fortran);
- диаграммно-графовое – визуальность (Delphi);
- сценарное – одноуровневое, поверхностное (Tcl);
- операционное – управление заданиями (JCL);
- языки действий – управление акторами (Actor);
- событийно-ориентированное – обработчики событий (PL/1).

Постановка задач базовой ООП основана на контроле доступа к иерархии классов объектов с работоспособными методами решения задач некоторой предметной области. Требуется без лишних трудозатрат уточнить эту иерархию, чтобы приспособить её к решению новых задач этой области, её расширения или

перехода к подобной. Производные ООП дают разнообразные конкретизации понятия «класс объектов» (S) и механизмов их размещения в памяти (M):

- аспектно-ориентированное – разделение слоёв (AspectJ);
- обобщённое (генеративное) – с абстрактными типами данных в роли классов (Ada);
- прототипное – без иерархии классов (Self);
- табличная – с таблицами в роли классов (Falcom);
- проблемно-ориентированное – синтаксическая надстройка (AutoLisp, Perl, Verilog, VHDL, SQL);
- протоколо-ориентированное – интерфейсы (Swift);
- компонентно-ориентированное – по архитектуре (Visualbasic);
- субъектно-ориентированное – объекты с поведением (Smalltalk);
- обмен сообщениями - (Falcon);
- событийно-ориентированное – (TypeScript);
- рефлексивно-ориентированное – (Dart).

Задачи базовой ФП обычно ориентированы на известную предметную область, в рамках которой следует выбрать символьное представление данных и отладить систему универсальных функций, пригодных для не чрезмерно трудоёмкого создания прототипов решения новых задач из этой области. Производные ФП представляют вариации в методах организации вычислений (E) и структурирования данных (S):

- чистое – методика снижения стартового барьера отказом от побочных эффектов и явного управления (PureLisp и Haskell без монад);
- динамическое – изменения по ходу вычислений (Lisp, Python);
- ленивые вычисления – откладывание выполнения действий (Hope);
- мемоизация – хранение результатов (Setl);
- комбинаторное – сведение к функциям (FP);
- аппликативное – подгрузка любого базиса (Lisp);
- символьные вычисления – математические формулы (Reduce);
- гомоиконное – подобие структур данных и программы (TRAC);
- полиморфное – классы объектов (CLOS);
- правила переписывания – одноуровневая подстановка данных (РЕФАЛ);

- метапрограммирование – конструирование программ (Lisp, РЕФАЛ);
- продолжение – передачи управления (Scheme);
- алгебраическое – выделение семантических систем (CoQ);
- реактивное – обработчики ситуаций (Scala);
- экспериментальное – проявление знаний (Lisp);
- стек-ориентированное (Forth).

Решения задач базовой ЛП исходят из заданной коллекции фактов и отношений, характеризующей актуальную задачу, пока не имеющую ни эффективного решения, ни полного описания. Надо привести эту коллекцию, возможно пополнить, к форме, демонстрирующей возможность ответов на практичные запросы относительно данной задачи. Производные ЛП используют разные подходы к смягчению зависимости получения результатов от избыточного или недостаточного детерминизма (C) и разные формы представления вычисляемых выражений (E):

- откаты-бэктрекинг – перебор вариантов (Snobol);
- недетерминированное – без преждевременного упорядочения (Prolog);
- вопрос-ответное – задачи поиска (NLP, Eliza);
- индуктивное – вывод решения (MIS, Prolog);
- естественно-языковое – запросы на подмножестве языка (MicroPlanner);
- программирование в ограничениях – границы определённости (Prolog);
- контрактное – задание спецификаций (D);
- предикатное – сведение к предикатам (P).

Существует заметное число **комбинированных** ПП, объединяющих достоинства двух-трёх ПП для решения разнотипных подзадач, поддерживаемых и **мультипарадигмальными** ЯП (Lisp 1.5, Planner, Merlin, F#, C#, Scala и др.). Происходящее в настоящее время проявление ПП в области улучшения ПО, оперирования устройствами, организации ПВ и обработки больших данных вероятно повлечёт переход ряда производных ПП в новые позиции парадигмального пространства.

Можно обратить внимание, что ЛП и ФП нередко рассматривают как декларативные, учитывая их приспособленность к работе на предварительных фазах определения постановки решаемой задачи. Более естественно выделять **дополнительные формы**, такие как декларативность, абстракции, языки спецификаций и др., преимущественно решающие задачи типа «строительных лесов», т. е. в центре внимания таких форм не альтернативное, противопоставляемое представление средств и методов программирования, а задание границ поведения программ, выделение удобных для практики порождаемых процессов. Например, декларативное представление типов данных присутствует во многих ЯП. Можно сказать, что дополнительные формы представляют собой методы наложения ограничительных условий на функционирование программы. Любая парадигма программирования может быть дополнена такими ограничительными условиями[7]. Большинство ограничительных условий теряет смысл по завершении отладки программы.

Ещё один важный ряд **вспомогательных** парадигм – это парадигмы области приложения программ, прикладные парадигмы, своевременный учёт которых обеспечивает успех результатов программирования. Зависимость от таких парадигм усложняет постановки задач для ЛП и ООП, что, возможно, побуждает специалистов по обучению информатике считать ФП, ПВ и ИПП более фундаментальными, чем ЛП и ООП [8].

Встречаются **отвергнутые** ПП, не получившие признания программистским сообществом вопреки подтверждающим экспериментам, дающие материал для исследования границ общей парадигмы программирования, наряду с **эзотерическими** ПП, изобретение которых можно рассматривать как исследование возможностей представлять и распознавать информацию в стиле создания и расшифровки ребусов.

ПЕРСПЕКТИВНО-СТРАТЕГИЧЕСКИЕ ПАРАДИГМЫ (ПСП)

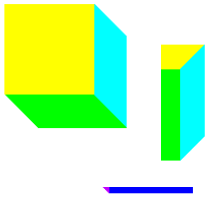
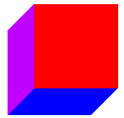
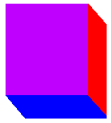
Редко упоминаются достаточно важные системообразующие **перспективно-стратегические** ПП, существование которых нацелено на общий прогресс ИТ, решение собственных задач программирования и развитие информатики как

[7] Математики говорят: «Знание границ закономерности освобождает от знания самой закономерности».

профессии. Такие ПП не сводимы к комбинации базовых и прикладных ПП, обладающих сторонней сферой приложения. Постановки задач здесь нацелены не только на улучшение качества результатов программирования, но преимущественно на стратегическую экстраполяцию перспектив повышения квалификации программистского корпуса, эстафету знания в области информатики и совершенствование технологий программирования, включая ИТ. Эксплуатационная прагматика таких ПП, в отличие от имеющих внешний полигон парадигм прикладного программирования, требует создания специальной программистской обстановки, включая элементы образовательной системы и комплекс мероприятий по обмену опытом с целью проявления и накопления имплицитных знаний, вербализация которых в программировании достаточно трудоёмка и слишком протяжённая по времени, т. к. требует «созревания головы». История ПП показывает, что появление наименований ПП обычно отстаёт от создания инструментальной поддержки примерно на 10–20 лет. Результативность специальной программистской обстановки ярко показана Ершовскими школами юных программистов, ежегодно проводимыми ИСИ СО РАН.

Следует отметить сравнительно сложившиеся ПСП, дополняющие ряд классических основных ПП. Прежде всего это фундаментальные ПП: алгоритмика и схематология, теоретическое (Standard Pascal, C-light), доказательное (CoQ) и верификационное программирования (Isabelle, PVS, LTL). Далее формируются образовательные ПП: учебное (PureLisp, Logo, Elm, Робик, Рапира, Oz), олимпиадное – спортивное (Pascal, C, Python), непрофессиональное (Basic) и экспериментальное программирования (Lisp). Производственное значение имеют технологичные ПП: оптимизационное и трансформационое программирование (работы Л. Ломбарди, Ё. Фатамуры, А.П. Ершова), любительское, отладочно-тестирующее и свободно-распространяемое программное обеспечение (GNU). Более детальная характеристика ПСП не входит в задачу данной статьи, что не мешает суммарно представить сводку ПП (Таблица 18), на которой символизировано, что общее пространство, составленное из ряда разных категорий ПП, можно рассматривать как грани куба, одна из которых выполняет роль ведущей, остальные интуитивно проявляются по мере развития постановки решаемой задачи.

Таблица 18. Парадигмы программирования (ПП)

Парадигма программирования					
	***Основные практические ПП***			***Перспективно стратегические ПП***	
Базовые императивно-процедурное функциональное логическое объектно-ориентированное	**Системно расширяющие** системное операционные системы системы управления БД многопроцессорные комплексы	**Неограниченные** большеобъёмные данные дистанционный доступ параллельное суперкомпьютерное (высокопроизводительное)	**Технологичные** оптимизационное трансформационое (проекции) отладочное и тестирование свободно-распространяемое	**Образовательные** учебное олимпиадное непрофессиональное экспериментальное (любительское)	**Фундаментальные** алгоритмическое теоретическое (схематология) доказательное верификационное
					

ЗАКЛЮЧЕНИЕ

Предложенную систематизацию ПП можно использовать при оценке сложности и трудоёмкости программирования, особенно если дополнить более ясным разделением требований к постановкам задач по сферам применения на академические и производственные, а по уровню изученности – на точные, развиваемые и усложнённые трудно удостоверяемыми требованиями. **Основные** (базовые, расширяющие и неограниченные) ПП можно различать по упорядочению категорий семантических систем и обрабатываемых данных, **производные** – по отличию от исходной парадигмы, **вспомогательные** – по областям приложения, **комбинированные** – по составляющим парадигмам. Любая парадигма программирования может быть обогащена **дополнительными** формами представления ограничительных условий на функционирование программ и вспомогательными прикладными ПП приведения к конкретной сфере приложения.

В качестве близких следует указать работы [6–8]. Е.М. Лаврищева представила достаточно полных обзор ПП, имеющих значение для технологий программирования [7], П. Ван Рой проанализировал более 30-ти ПП, преимущественно комбинированных, и представил схему их взаимосвязей, цитируемую в Википедиях [8], а П. Вегнер выполнил весьма серьёзный анализ ООП, методов поддержки этой парадигмы и её сравнение с другими классическими ПП [6].

Работа выполнена при поддержке Российского фонда фундаментальных исследований, проект № 18-07-01048-а.

СПИСОК ЛИТЕРАТУРЫ

1. https://www.levenez.com/lang/ – Диаграмма, представляющая хронологию появления и наследования многих ЯП.

2. http://progopedia.ru/. Сайт с описаниями 171 языка и 31 парадигмы.

3. *Лавров С.С.* Методы задания семантики языков программирования // Программирование, 1978. № 6. С. 3–10.

4. *Бентли Д.* Жемчужины творчества программистов. М.: Издательство «Радио и связь»: Редакция переводной литературы, 1990. 217 с.

5. *Городняя Л.В.* О представлении результатов анализа языков и систем программирования. Научный сервис в сети Интернет: труды XX Всероссийской научной конференции (17–22 сентября 2018 г., г. Новороссийск). М.: ИПМ им. М.В. Келдыша, 2018.

6. *Peter Wegner.* Concepts and paradigms of object-oriented programming. SIGPLAN OOPS Mess. 1, 1 (August 1990). P. 7–87. https://pdfs.semanticscholar. DOI: http://dx.doi.org/10.1145/

7. *Лаврищева Е.М.* Программная инженерия и технологии программирования сложных систем. Учебник для вузов. М., 2018. 432 с.

8. *Peter Van Roy.* Диаграмма с результатами сравнения более 30-ти парадигм программирования. https://www.info.ucl.ac.be/~pvr/paradigmsDIAGRAMeng108.pdf

---

ON SYSTEMATIZATION OF PROGRAMMING PARADIGMS BY DECISION-MAKING PRIORITIES

L. V. Gorodnyaya

A.P. Ershov Institute of Informatics Systems, Siberian Branch of the Russian Academy of Sciences, Novosibirsk State University, Novosibirsk

lidvas@gmail.com

Abstract

The report is devoted to the analysis of the method of comparison of programming languages, convenient for assessing the expressive power of languages and the complexity of the programming systems. The method is adapted to substantiate practical, objective criteria of program decomposition, which can be considered as an approach to solving the problem of factorization of very complicated definitions of programming languages and their support systems. The article presents the results of the

analysis of the most well-known programming paradigms and outlines an approach to navigation in the modern expanding space of programming languages, based on the classification of paradigms on the peculiarities of problem statements and semantic characteristics of programming languages and systems with an emphasis on the criteria for the quality of programs and priorities in decision-making in their implementation.

Keywords*: definition of programming languages, definition of programming languages, programming paradigms, definition decomposition criteria, semantic systems*

REFERENCES

1. https://www.levenez.com/lang/ – Chart representing the history of the emergence and inheritance of many programming languages.

2. http://progopedia.ru/. Website with descriptions 171 languages and 31 paradigms.

3. *Lavrov S.S.* Methods of definition the semantics of programming languages // Programming. 1978. No 6. P. 3–10.

4. *Bentley D.* The Pearls of creativity of programmers. Moscow: publishing House "Radio and communication": Edition of translated literature, 1990. 217 p.

5. *Gorodnyaya L.V.* On the presentation of the results of the languages and programming systems analysis. Scientific service on the Internet: proceedings of the XX All-Russian scientific conference (17–22 September 2018, Novorossiysk). Moscow: IPM im. M.V. Keldysh, 2018.

6. *Peter Wegner.* Concepts and paradigms of object-oriented programming. SIGPLAN OOPS Mess. 1, 1 (August 1990). P. 7–87. https://pdfs.semanticscholar. DOI: http://dx.doi.org/10.1145/.

7. *Lavrishcheva E.M.* Software engineering and programming technologies of complex systems. Textbook for high schools. Moscow, 2018. 432 p.

8. *Peter Van Roy.* Chart with the results of comparison of more than 30 programming paradigms. https://www.info.ucl.ac.be/~pvr/paradigmsDIAGRAMeng108. pdf.

СВЕДЕНИЯ ОБ АВТОРЕ

ГОРОДНЯЯ Лидия Васильевна – старший научный сотрудник Института систем информатики имени акад. Андрея Петровича Ершова СО РАН, доцент Новосибирского государственного университета, специалист в области системного программирования и образовательной информатики.

Lidia Vasiljevna GORODNYAYA – Senior Researcher of A.P. Ershov Institute of Informatics Systems, Siberian Branch of the Russian Academy of Sciences, Associate Professor of Novosibirsk State University, a specialist in system programming and educational informatics.

email: lidvas@gmail.com

Материал поступил в редакцию 15 ноября 2019 года