ИПМ им.М.В.Келдыша РАН

Абрау-2019 • Труды конференции

Л.В. Городняя

Методика парадигмального анализа языков и систем программирования

Рекомендуемая форма библиографической ссылки

Городняя Л.В. Методика парадигмального анализа языков и систем программирования // Научный сервис в сети Интернет: труды XXI Всероссийской научной конференции (23-28 сентября 2019 г., г. Новороссийск). — М.: ИПМ им. М.В.Келдыша, 2019. — С. 262-277. — URL: http://keldysh.ru/abrau/2019/theses/03.pdf doi:10.20948/abrau-2019-03

Размещена также презентация к докладу

Методика парадигмального анализа языков и систем программирования

Л.В. Городняя

Институт систем информатики СО РАН, НГУ

Аннотация. Цель статьи — описание методики сравнения языков программирования, удобной для оценки выразительной силы языков и трудоёмкости реализации систем программирования. Методика приспособлена 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. Кроме того, представлены результаты анализа наиболее известных парадигм программирования и намечен подход к навигации в современном расширяющемся пространстве языков программирования, основанный на классификации парадигм по особенностям постановок задач и семантической характеристики языков и систем программирования с акцентом на критерии качества программ и приоритеты в принятии решений при их реализации.

Ключевые слова: определение языков программирования, парадигмы программирования, критерии декомпозиции определений, семантические системы

Method of paradigmatic analysis of programming languages and systems

L.V.Gorodnyaya

A.P. Ershov Institute of Informatics Systems (IIS)

Abstract. The purpose of the article is to describe 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. In addition, 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, programming paradigms, definition decomposition criteria, semantic systems

Описания современных языков программирования (ЯП) обычно содержат список из 5-10 предшественников и ряд парадигм программирования (ПП), поддержанных языком [1,2]. Такая характеристика как правило не показывает, какие свойства ПП и черты ЯП фактически унаследованы определяемым языком и какие особенности являются действительно новыми (см. Таблицы 1,2)[1]. Для реальной оценки уровня новизны и практичности ЯП может быть полезной коллекция постановок задач с примерами их решения на разных языках в рамках разных 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 поддерживают механизмы ввода-вывода на уровне библиотечных вызовов, такие примеры слабо характеризуют язык.

В данной статье рассматривается методика представления 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 постановки задачи и используемого для её решения инструмента. На основе такой методики 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 парадигм программирования в рамках ЯП высокого уровня без экскурса в языки низкого и сверх высокого уровней или более широкое пространство парадигм, особенно новых, ещё не получивших поддержки в достаточно известных языках программирования и признания в виде примеров отлаженных программ. На будущее оставлен анализ языков организации параллельных вычислений (ПВ), заслуживающих не менее двух парадигм, и DSL-языков, вероятно знаменующих переход прикладного программирования на качественно новый уровень.

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

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

1. 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. Таблица 1).

Таблица 1

ЯП XXI века *(все м*ультипарадигмальны*е)*

Год	*ЯП*	*Предшественники*[2]	*Поддержанные парадигмы*
2018	Dart	Java, JavaScript, CoffeeScript, Go	объектно-ориентированный каркас веб-приложений сценарный язык императивный рефлективный функциональный
2014	Swift	Objective-C, C++, Java, Rust, Scala, Python, Ruby, Smalltalk, Groovy, D, LLVM	протоколо-ориентированный объектно-ориентированный функциональный императивный
2012	Rust	Alef, C++, Camlp4, Common Lisp, Erlang, Haskell, Hermes, Limbo, Napier, Napier88, Scheme, Newsqueak, NIL, Sather, OCaml, Standard ML, Cyclone, Swift, C#, Ruby	параллельный функциональный императивный структурный системный процедурный свободное программное обеспечение
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,	объектно-ориентированный обобщённый процедурный

[2] Сохранена лексика источников.

		Модула-3, Smalltalk	функциональный событийный рефлективный

Интересно отметить, что парадигмальная характеристика многих долгоживущих ЯП исторически претерпела заметные изменения (см. Таблица 2). Можно сказать, что отдельные парадигмы проявились постепенно, по мере накопления опыта решения определённых классов задач.

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 обобщённое объектно-ориентированный	**ИПП — императивно-процедурное** ALGOL 58, BASIC, C, Chapel, CMS-2, Fortress, PL/I, PACT I, MUMPS, IDL, Ratfor
1958	Lisp	эксперименальный 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++, Bourne 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	**СП — структурное**

[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.

		структурный	Ada, Component Pascal, Modula-2, Java, Go, Oberon, Object Pascal, Oxygene, Seed7, VHD, Structured text

Парадигма программирования как образ мышления связана с компромиссом между особенностями решаемых задач, методами их решения в форме программ, принятым в ПП критерием качества программ и приоритетами принятия решений в процессе программирования. Мультипарадигмальность 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 решаемых задач. Для показа особенностей ПП удобно выделять концептуальные монопарадигмальные ЯП и приводить критерии успешного применения ПП с оценкой результатов на примерах программ, подтверждающих признание практикой программирования [5].

2. Семантические системы базовых парадигм

Рассматривая любые семантические системы, важно отметить разницу в характере выполнения функций таких систем в разных контекстах. 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 моделях семантических систем по разному реализуемых схем функций F. Исторически на уровне аппаратуры такие модели систем обладали кумулятивным эффектом в порядке «DEMCS» (см. Таблица 3) — представление чисел, арифмометр, калькулятор, дифференциальный вычислитель, компьютер, причём, аппаратные подсистемы могут взаимодействовать каждая с каждой.

Таблица 3

Ряд моделей 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

Парадигмы программирования можно отличать по приоритетам выбора решений при определении моделей семантических систем в процессе программирования, отмечая парадигмальные различия общих понятий в каждой модели, зависящие от критериев качества программ (См. Таблицы 4-7). 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.

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. Таблица 4).

Таблица 4

Парадигмальная шкала семантики 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 предметной 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. Таблица 5).

Таблица 5

Парадигмальная шкала семантики 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 произвольно 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 произволном 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. Таблица 6).

Таблица 6

Парадигмальная шкала семантики 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

Ведущий критерий качества программ в ООП – сопоставимость иерархии классов объектов с иерархией понятий в области приложения программ. 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. Таблица 7).

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

Таблица 7

Парадигмальная шкала семантики 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

Таким образом, видны отличия в схемах определения функций для разных моделей 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.

3. 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 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);
- диаграммно-графовое — визуальность (Delphi);
- сценарное — одноуровневое, поверхностное (Tcl);
- операционное — управление заданиями (JCL);
- языки действий — управление акторами (Actor);
- векторно-ориентированное — обработка массивов (APL);
- событийно-ориентированное — обработчики событий (PL/1);
- синхронизация процессов над общей памятью (OpenMP).

Постановка задач 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, Lex/Yacc, Perl, Verilog, VHDL, SQL);
- субъектно-ориентированное — объекты с поведением (Smalltalk);
- сервис-ориентированное — выделение обслуживания (XML).

Задачи 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);
- 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);
- продолжение — передачи управления (Scheme);
- алгебраическое — выделение семантических систем (CoQ);
- инсерционное — динамические вставки-замены (SETL);
- грамматико-ориентированная обработка (BNF);
- реактивное — обработчики ситуаций (Scala);
- стек-ориентированное (Forth).

Решения задач 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);
- представление знаний — привлечение экспертов (GPS, RDF, OWL);
- вопрос-ответное — задачи поиска (NLP, Eliza);
- индуктивное — вывод решения (MIS, Prolog);
- естественно-языковое — запросы на подмножестве языка (MicroPlanner);
- программирование в ограничениях — границы определённости (Prolog).

Кроме того, 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 ИС, оперирования устройствами, организации ПВ и обработки больших данных вероятно повлечёт переход ряда производных ПП в новые позиции парадигмального пространства.

Нередко вместе с ПП упоминают **дополнительные** парадигмы, такие как декларативное, математическое, языки спецификаций и др., преимущественно решающие задачи типа «строительных лесов», т. е. в центре внимания таких парадигм не альтернативное, противопоставляемое представление средств и методов программирования, а задание границ поведения программ, выделение удобных для практики порождаемых процессов. Можно сказать, что дополнительные парадигмы — это 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 ряд **вспомогательных** парадигм — это парадигмы области приложения программ, своевременный учёт которых обеспечивает успех результатов программирования. Зависимость от таких парадигм характерна для ЛП и ООП, что побуждает некоторых специалистов 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.

3. Системообразующие парадигмы (СОП)

Редко упоминаются достаточно важные инструментальные, **системообразующие** ПП, существование которых нацелено на общий прогресс

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

ИТ, решение собственных задач программирования и развитие информатики как профессии. Такие ПП не сводимы к комбинации базовых ПП, обладающих сторонней сферой приложения. Постановки задач здесь нацелены на улучшение качества результатов программирования, повышение квалификации программистского корпуса, эстафету знания в области информатики и совершенствование технологий программирования, включая ИТ. Эксплуатационная прагматика таких ПП, в отличие от имеющий 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 протяжённа 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 системообразующие парадигмы (СОП) :

- системное программирование: системы программирования, базы данных, операционные системы (VDM, BNF, Unix, Lex/YACC, Bliss, ЯРМО, C, SQL);
- оптимизационное программирование (Внутренний язык системы БЕТА);
- компонентное программирование (Com/Dcom, Corba, SOAP, **.**Net);
- экспериментальное программирование (Lisp);
- трансформационое программирование (работы Л.Ломбарди, Ё.Фатамуры, А.П.Ершова);
- теоретическое программирование (Standard Pascal, C-light);
- метапрограммирование (Рефал);
- доказательное программирование (CoQ);
- верификационное программирование (Isabelle, PVS, LTL);
- учебное программирование (PureLisp, Logo, Elm, Робик, Рапира, Oz);
- олимпиадное программирование — спортивное (Pascal [8], C, Python);
- непрофессиональное программирование (Basic [9]).

Кроме разницы между научно-исследовательскими (академическими) и производственными задачами, области приложения существенно различаются по следующим направлениям:

[8] Язык Pascal, созданный в качестве учебного языка, в течение десятилетия работал как производственный язык и дал материал для научных исследований.

[9] Язык Basic, созданный для непрофессионального программирования, заметное время при переходе на микропроцессоры работал как основной язык прикладного программирования.

- точные, почти формализованные, постановки задач, решаемых с помощью конечного автомата (ИПП, ФП);

- развивающиеся, не вполне формализуемые, постановки задач, требующие для решения расширяемого автомата (ЛП, ООП);

- более общие задачи функционирования многопроцессорных конфигураций, решаемые как комплексы взаимодействующих процессов или пространства автоматически выбираемых методов (ПВ, СОП);

- усложнённые задачи, связанные с трудно удостоверяемыми критериями качества обработки очень больших массивов, обеспечения надёжности и безопасности информационного обслуживания (СОП).

Заключение.

Предложенную методику парадигмального анализа языков и систем программирования 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. **Базовые** парадигмы программирования можно различать по упорядочению основных моделей семантических систем, **производные** — по отличию от базовой парадигмы, **комбинированные** — по составляющим парадигмам. Кроме того имеются **системообразующие** ПП, не сводимые к простым проекциям или сочетаниям базовых ПП. Любая парадигма программирования может быть обогащена **дополнительными** парадигмами представления ограничительных условий на функционирование программ. По этой причине они не могут быть противопоставлены собственно ПП.

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.

Предметно-ориентированные DSL-языки заслуживают отдельного рассмотрения 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.

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].

Есть основания выделять ПП, нацеленные на обеспечение обратной связи при работе с устройствами и сетями, на поверхностно-интерфейсный стиль ИТ и на поддержку суперкомпьютерных процессов.

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.

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

Литература

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. N 6. С. 3-10.
4. Бентли Д. Жемчужины творчества программистов. Москва: 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. 1990. 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 г. 432 с.
8. Peter Van Roy. Диаграмма с результатами сравнения более 30-ти парадигм программирования.
https://www.info.ucl.ac.be/~pvr/paradigmsDIAGRAMeng108.pdf .

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. N 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. 1990. 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 г. 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 .