УДК 004.43 (042.4)

ПРОЕКТ ИНФОРМАЦИОННОГО СТЕНДА «ПРИЗМА» ДЛЯ ПРЕДСТАВЛЕНИЯ ИЗМЕРИМЫХ ХАРАКТЕРИСТИК ЯЗЫКОВ И СИСТЕМ ПРОГРАММИРОВАНИЯ

Городняя Лидия Васильевна
К.ф.-м.н., доцент, с.н.с.
Институт систем информатики СО РАН,
630090 г. Новосибирск, проспект Академика Лаврентьева, 6, e-mail: gorod@iis.nsk.su
Демидов Сергей Евгеньевич
Магистрант, НГУ, 630090, Новосибирск, ул. Пирогова, 1, e-mail: sersh_96@mail.ru
Кириченко Михаил Дмитриевич
Магистрант, НГУ, 630090, Новосибирск, ул. Пирогова, 1, e-mail: m.kirichenko@g.nsu.ru
Ткаченко Денис Дмитриевич
Студент, НГУ, 630090, Новосибирск, ул. Пирогова, 1, e-mail: d.panfilov@g.nsu.ru

Аннотация. Статья посвящена вопросам, возникающим в связи с проблемой измерения характеристик языков и систем программирования, влияющих на трудоёмкость разработки программного обеспечения и производительности программных приложений. Парадигмальные модели языков и систем программирования могут быть полезны при систематизации языков программирования, оценке их сходства и различия, что позволяет строить лаконичные определения относительно таких моделей. Это даёт возможность стратификации представления особенностей семантики языков программирования на автономно развиваемые компоненты в процессе пошаговой разработки экспериментальных систем программирования и формирования схем изучения и преподавания системного программирования.

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

Ключевые слова: систематизации языков программирования, парадигмальные модели, автономно развиваемые компоненты, схемы преподавания системного программирования, лаконичные определения.

..

Цитирование: Городняя Л.В., Демидов С.Е., Кириченко М. Д., Ткаченко Д. Д. Проект информационного стенда «ПРИЗМА» для представления измеримых характеристик языков и систем программирования // Информационные и математические технологии в науке и управлении. 2020 № 1 (17) С. 105– 119. DOI: 10.38028/ESI.2020.17.1.008

Введение. Информационный стенд ПРИЗМА (**пр**ограммирование **изм**ерений) предназначен для представления экспертных оценок возможностей языков программирования (ЯП) и результатов измерения и анализа эксплуатационных характеристик, написанных на разных языках программирования, отлаживаемых на конкретных системах программирования (СП) при определённой конфигурации оборудования. В данной статье предпринята попытка создать методику измерения характеристик языков и систем программирования в форме,

позволяющей ориентироваться в современном пространстве средств и методов, определяющих дальнейшее развитие информационных технологий (ИТ), используя наблюдения практиков, заметивших значительный объём интуитивной работы программиста и большое число параметров при оценке качества реализации СП [1, 2, 9], а также идеи понятийной сложности [8], вертикального слоения программ [11] и семантической системы [10]. Понятие «Парадигма программирования» здесь конкретизирует формулировку П.Вегнера «*правила классификации языков программирования в соответствии с некоторыми **условиями**, которые могут быть проверены*» [13] определением **условий** как приоритетов принятия решений, обусловленных целями решаемой задачи и требованиями к её решению [3].

Следует отметить, что жаргон современного практического программирования понятие «язык программирования» использует как «входной язык (расширенное подмножество ЯП) типовой системы программирования, функционирующей на базе определённой конфигурации оборудования». Разница заключается в том, что СП обычно сопровождает реализацию ЯП расширяемым комплектом библиотечных модулей. В результате происходит сглаживание видимых различий между языками и системами программирования (ЯСП). Кроме того, прямые измерения трудоёмкости программирования и производительности программ почти не отражают зависимости от принятых программистом решений и выбора конструкций ЯП. Таким образом, существует проблема создания методики, позволяющей выяснять такие зависимости совмещением прямых измерений с результатами экспертных оценок особенностей ЯСП.

В разных источниках упоминаются от 20-ти до 70-ти парадигм программирования (ПП) [12-15], список которых видоизменяется и расширяется в зависимости от актуальности тех или иных проблем программирования и моды на особенности популярных ИТ. Бум языкотворчества в области проблемно-ориентированных ЯП, знаменующий переход практики программирования от накопления опыта на уровне библиотечных модулей к накоплению подъязыков, показывает важность парадигмальных различий в определениях языков при выборе инструментов для практических работ. Ещё одна тенденция видна по восстановлению интереса к созданию монопарадигмальных ЯП, более удобных для исследования и изучения. Разработка методов анализа определений ЯСП с целью установления их парадигмальной характеристики может дать основу для выработки рекомендаций по выбору инструментальных средств при создании новых программных систем и развитии ИТ.

1. Представление результатов анализа ЯП. Стенд ПРИЗМА предлагает изучение описаний языков программирования (ЯП) сопровождать парадигмально-семантической декомпозицией, результаты которой представляются понятийными таблицами, пример приведен в таблице 1, пояснения к заполнению клеток таблицы даны в [4]. Из такой таблицы можно вывести матрицу численных характеристик понятийной сложности ЯП (см. таблицу 2) и построить визуальную диаграмму, зрительно показывающую различия в семантический структуре сравниваемых ЯП, достаточную для навигации в общем корпусе ЯП (см. таблицу 3). Точки такой диаграммы реализуются как указатели на позиции обучающего ряда примеров, иллюстрирующих смысл понятий ЯП (см. таблицу 4). Для мультипарадигмальных ЯП можно представить несколько понятийных таблиц по числу поддержанных парадигм. Наполнение

понятийных таблиц разными экспертами могут отличаться. Ряд примеров может отражать методику обучения.

Таблица 1. Понятийная матрица определения языка Pure Lisp (ядро) [1]

№ стр	№ столбца	V	**E**	**M**	C	**S**
	Виды функций	V	**F: V* → V**	**AL : Atom → V**	**(F → {NIL, T})***	**A ↔ K**
E	Ядро	•NIL ; и атомы	•eq: V V→{NIL,T}	•AL = (… (NIL . NIL) (T . T))	•eval:Sexpr AL→ V •quote	•cons: V1 V2 → (V1 . V2) •car: (V1 . V2) →V1 •cdr: (V1 V2) →V2
M	Специальные функции	•Sexpr: (V . V) •(V . . .) ; список	•(lambda (x1 x2 ...) form) •(label FN F)	•pairlis X Y → AL •assoc: X AL → V	•((lambda (x1 x2 ...) form) v1 v2 …) •((label FN F) …)	•list ; Список •(F v1 v2 . . .) ; Форма
C	Границы	•NIL •«Нет переменной» •«Нет функции»	•atom:V→{NIL, T} •apply: F V* AL → V •ERROR	•null •Свободные переменные •Типы значений	•cond •FUNCTION	•Строки •Clozure = Funarg
S	Общность Практичность	•Числа •Строки	•evlis : (form*) → V* •evcon : ((form form)*) → V •+ - * /	•Псевдо-функции •свойства	•map •reduce •lazy	•READ •PRINT

1 V – value, E- evaluation, M – memory, C – control, S - srtucture

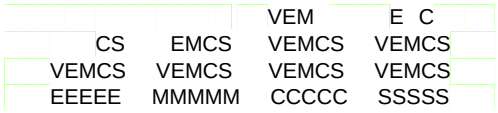
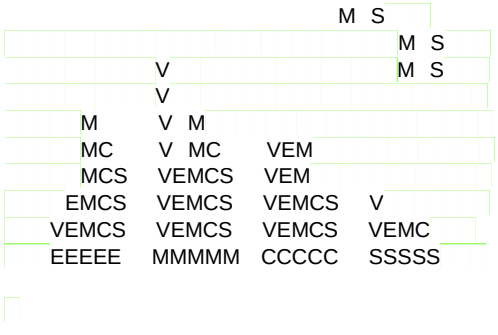
Такие понятийные матрицы достаточны для представления результатов анализа ЯП и грубой оценки сложности применения при программировании, например, по числу различимых конструкций в каждой клетке понятийной матрицы (таблица 2).

Таблица 2. Представление результатов сравнения понятийной сложности определений языков Pure Lisp и Pascal.

Lisp						**Pascal**					
	V	E	M	C	S		V	E	M	C	S
E	1	1	1	2	3	E	1	2	5	4	3
M	2	2	2	2	2	M	7	2	5	4	3
C	3+	3	3	2	2	C	3+	4	3	2	3
S	2	3	2	3	2	S	2	1	1 /**3**	1	0 /**3**

Переход к содержательно эквивалентной графической диаграмме позволяет практически мгновенно сравнивать сложность ЯП на уровне системы понятий в терминах категорий семантических систем (таблица 3).

Таблица 3. Визуальные формы представления результатов сравнения сложности ЯП.

ЯП	Численная оценка понятийной сложности	Графическая диаграмма для быстрого сравнения	Поддержанные парадигмы
Lisp 42	(E (1 1 1 2 3) M (1 2 2 2 2) C (3 3 3 2 2) S (2 3 2 3 2))		экспериментальный функциональный объектно-ориентированный процедурный рефлективный мета-программирование
Pascal 56	(E (1 2 5 4 3) M (7 2 5 4 3) C (3 4 3 2 3) S (2 1 **1** 1 **0**)) + em3, ss3		императивный структурный

На графической диаграмме видно резкое различие языков, неоднородность наполнения категорий семантических систем, изменение числа понятий.

Для более точной оценки ЯСП и привлечения измеримых характеристик используются комплекты сопоставимых фрагментов программ на оцениваемых ЯП в форме, приспособленной для прямого эксперимента с СП, поддерживающих изучаемые ЯП. Примеры накапливаются как упорядоченный по сложности их усвоения обучающий ряд, вид которого приведён в таблице 4.

Таблица 4. Начало ряда примеров программ (примеры ФП: Pure Lisp – чистые вычисления)

Индекс	Понятие	Примеры	Примечания
	Любую информацию можно представить символами. Информационная обработка может быть сведена к выражениям из функций с использованием небольшого числа операций над символами		Исходная гипотеза
EVa	Минимальные значения	NIL	= ( ) = пустой список и «ложь»
		T и другие атомы	Атомы без начального значения
EEa	EQ (равенство)	(EQ NIL NIL)	«истина»
		(EQ T T)	Если T имеет значение, то «истина»
		(EQ NIL T)	«ложь» или «неопределено»
ESa	CONS	(CONS NIL NIL)	Пара (NIL . NIL) = (NIL)
		(CONS x y)	Пара (x . y), **если** x и y имеют значение
ESb	CAR	(CAR (CONS x y))	x - Левый элемент пары (x . y)
ESc	CDR	(CDR (CONS x y))	y - Правый элемент пары (x . y)
EMa	Ассоциативный список	((T . T) (NIL . NIL))	Содержимое памяти, показывающее, что NIL и T означают сами себя, т.к. связаны сами с собой
ECa	EVAL (Вычисление)	(EVAL NIL) (EVAL T) (EVAL (CONS T NIL)) (EVAL A)	NIL T (T . NIL) = (T) **Если** A вычислимо, то его значение, **иначе** «неопределено»
ECb	QUOTE (Блокировка) ' - эквивалент	(QUOTE A) (QUOTE (EVAL A)) (EVAL (QUOTE A))	A (EVAL A) A
		'A ' (EVAL A) (EVAL ' A)	; эквивалент (QUOTE A) ; эквивалент (QUOTE (EVAL A)) ; эквивалент (EVAL (QUOTE A))

Каждый пример иллюстрирует особенности использования отдельного понятия. Для иллюстрации некоторых понятий может требоваться несколько фрагментов. Последовательность элементов ряда соответствует порядку изучения понятий, подкреплённого компьютерной практикой. Каждых элемент содержит представление результата прогона фрагмента на СП и пояснение схемы применения таких фрагментов в

подходящих задачах. Число примеров, необходимых для понимания практики применения понятия в программах, можно рассматривать как оценку сложности изучения понятия. В зависимости от целей обучения над обучающим рядом можно строить маршруты или учебные траектории, отражающие особенности решаемых задач или сферы применения их решений.

2. Представление результатов сравнения ЯСП. Представление результатов сравнения возможностей ЯП и эксплуатационных характеристик СП на основе понятийных таблиц [3-6, 8, 9] и обучающих рядов выполняется техникой, подобной работе с многомерными неоднородными массивами. Можно сравнивать наполнение, оценку сложности или измеримые характеристики по отдельным клеткам, строкам, столбцам, диагоналям или произвольным композициям накопленных данных, позволяющих обосновывать выводы об особенностях средств решения разных задач. Более важно, что можно чётко выражать сходство и различие ЯП. Сравнивая пару ЯП1 и ЯП2, можно выделять общее (ЯП1 ∩ ЯП2) и две несимметричных разности (ЯП1 ⊀ ЯП2) или (ЯП1 ⊁ ЯП2). Для пары «предшественник-потомок» проявляется более ясное представление различий, возникающих при развитии ЯП.

Таблица 5. Результаты сравнения двух ЯП на уровне базовой семантики

№ позиции	Пояснение	1960 - Лисп	1970 - Паскаль	Комментарий = общее ≠ различие - потеряно + добавлено ! новое
Ядро	Видна разница в стартовом барьере при изучении ЯП	8: E (1 1 1 2 3)	15: E (1 2 5 4 3)	Число семантических систем
		CS VEMCS EEEEE	M MC MCS EMCS VEMCS EEEEE	Диаграммы доступа к примерам
EV	Самоопределяемые скаляры, их смысл не требует интерпретации, текстовое представление понятно без комментариев.	•NIL ; и атомы	•type = (a, b, . . .)	= общее — в обоих ЯП можно вводить свои различимые имена; ≠ различие — в Лиспе, кроме NIL, атомы изначально не определены, а в Паскале самоопределимые константы означают номер вхождения а перечнь; - потеряно — возможность определять смысл по ходу дела; + добавлено — чёткость смысла имён

EE	Операции над скалярами, они должны вырабатывать скаляры. При выходе за границы V могут его пополнять или вырабатывать сигнал неуспеха. Множество значений может быть пополнено представлениями формул.	eq: V V→{NIL,T}	• = <> • pred succ	В Паскале функций больше: = общее — можно сравнивать значения; ≠ различие — в Лиспе результат обычное значение NIL или отличное от него, а Паскале появляется специальный тип логических констант False, True; - потеряно – возможность любую операцию или функцию применять как предикат; + добавлено — контролируемая граница между вычислениями и сравнениями; - добавлено — средства использовать упорядочение значений, переходить к «соседям»
EM	Обработка памяти над таблицей адресов с соответствующим и им значениями. Адреса и таблица могут быть значениями или не рассматриваться как значения и использоваться неявно как сущности другой природы. Появляется именование значений и формул, что расширяет класс допустимых формул, обеспе-чивает много-кратное исполь-зование хранимых результатов и при водит к понятию «данное».	•AL = (… (NIL . NIL) (T . T))	•**Id** •var X : int; •X := 1; •label L; •L: a := 1;	= общее — можно связывать имена с представлением их смысла; 2≠ различие - Лисп не требует особых понятий для организации памяти, ассоциативный список AL – это просто обычная структура данных, а Паскаль требует не только понятия “распределение памяти под идентификаторы”, но они ещё разделены на метки и переменные, причём организацтия их хранения в языке не определена, зависит от реализации; - потеряно — принцип Фон Неймановской архитектуры: равноправие программ и данных; + добавлено — разделение структуры данных и структуры управления вычислениями

EC	Безусловное и условное (без «else») управление процессом вычислений, приоритеты в формулах вычисления и переход к очередному действию или по метке.	•eval:Sexpr AL→ V •quote	•(1*(2+X)) •S1 ; S2 •ключевые слова •goto L;	= общее - используется иерархия выражений; 10≠ различие – управление последовательностью вычислений в Паскале выполняется разными механизмами, выраженными с помощью ключевых слов, а в Лиспе всё или вычисляется или блокируется; ≠ различие –-Лисп-выражения - префиксные, а Паскаль-выражения – инфиксные; - потеряно – свобода от учёта приоритетов операций; + добавлено– наследование привычек к принятым в арифметических выражениях формам
ES	Конструирование последовательностей по принципу соседства и перебора слева направо дополняются возможностью возвратов или выбора любого элемента ради обратимости. Как правило это вектора или строки.	•cons: V1 V2 → (V1 **.** V2) •car: (V1 **.** V2) →V1 •cdr: (V1 . V2) →V2	•s: array [1 . . 9] of int; •s [5] := x; •x := s [5]	= общее – возможность доступа ко многим данным через одно обозначение или имя; ≠ различие – Лисп сводит структуры данных к парам, а Паскаль к блоку регистров, расположенных по соседству; - потеряно – автоматизация повторного использования памяти GC (сборка мусора) 3+ добавлено – распределение памяти соответственно описаниям имён; 2+ добавлено – возможность передач управления и присваивания по адресам имён и индексам векторов
Ядро	Оценка сходства-различия на уровне базовой семантики языков Лисп и Паскаль: **5= 15≠ 5- 7+ 0! (новых нет)**	В обоих ЯП представлены все категории семантических систем с вариациями по практике применения: 4 = совпадает выбор цели семантической системы; 14≠ различаются механизмы реализации 4- при переходе от Лиспа к Паскалю утрачены некоторые направления в пространстве решений для новых задач; 7+ при переходе от Лиспа к Паскалю расширено множество диагностических ситуаций.		

Lisp >> Pascal	= + 2n≠ + **4a+ + 8n!** 5+30+28 = 63	Разница в понятийной сложности определения Паскаля в сравнении с Лиспом

Ознакомление с описанием ЯП обычно требует от нескольких часов до нескольких дней. Использование понятийных матриц позволяет за 10-30 минут отбраковать описания ЯП, заведомо не представляющие интереса. Сравнение ЯСП требует несколько большего времени. Переход к численной характеристике понятийной матрицы позволяет отбраковку несравнимых ЯП выполнять за считанные минуты, но требует определённого напряжения на всматривание в запись чисел. Графическая диаграмма тех же самых чисел сравнивается практически мгновенно, что позволяет наладить методику экспертно-аналитической оценки возможностей и особенностей ЯСП. Для экспериментального исследования аналитики могут использовать свои и готовые примеры программ решения типовых задач на базе платформ, предоставляющих доступ к СП для большого числа языков (более 70-ти), дополненной доступом к реализации небольшого числа других языков.

Встречается более одного представления возможностей ЯП и ряд понятийных матриц, задающих его парадигмальную декомпозицию. Примеры представляют постановку задачи и/или программу её решения, что позволяет чётко оценить возможности ЯП на уровне шаблонов. Предпочтительны монопарадигмальные примеры с выделением конструкций, соответствующих элементам понятийной матрицы.

Формально численная характеристика эквивалентна графической диаграмме, длина столбика символизирует число, но различие диаграмм распознаётся заметно быстрее, Мгновенно видно, что сравниваемые ЯП отличаются друг от друга. Это не препятствует существованию отображений одного ЯП в другой. Разница между ними заключается в трудоёмкости представления программ решения разных типов задач и отладки представленных программ или в производительности и эффективности их реализации в СП. При анализе и сравнении ЯП можно учитывать предшественников и потомков, данные о которых доступны в большом числе источников. Такие данные желательно дополнить сведениями о разнице между сравниваемыми ЯП.

Конечно, для двух-трёх десятков ЯП не так уж трудно построить такие оценки. Для оперативной работы по анализу и сравнению сотен и тысяч современных ЯП, число которых уже превосходит десятки тысяч, нужна автоматизация даже столь простых преобразований и переход к представлению парадигмально семантических характеристик ЯСП в форме фрагментов программ с доступом к экспериментальным стендам, доступным уже для заметного числа ЯП.

Таблица 6. Сравнение фрагментов программ на уровне ядра языка

Индекс	Понятие	Примеры Паскаль	Примеры Лисп	Граница функциональной эквивалентности
EVa EVa	•type sym = (a, b, . . .)	type week =(sun,mon,tue,wed,thu,fri,sat); type valid_for_identifiers = 'a'..'z','A'..'Z','_','0'..'9';	NIL T ATOM (вс пн вт ср чт пт сб) 1a2	С точностью до алфавита

			2+3	
EEa EEa	a) = <>	ord (thu) = 4 thu <> sat	(EQ NIL NIL) (EQ T T) (EQ NIL T)	С точностью до совпадения типов данных
EEb	b) pred succ	Pred (mon) = Sun succ (tue) = wed ord(false)=0, ord(true)=1, false<true, pred(true)=false, succ(false)=true,	Допускают модель из асоциативного списка	С точностью до времени выполнения
Ema	a) Id	week week51 week_week	((T . T) (NIL . NIL)) ассоциативный список достаточен для моделирования всех видов работы с памятью	С точность до разницы между локальным и глобальным связыванием
EMb	b) var X : int;	var s: int;	Допускают модель из отдельного асоциативного списка, в котором связываются s и int;	С точностью до статического-динамического контроля
Emc EMa	c) X := 1;	int1 := -10; boolean6 := true;	модель из асоциативного списка, в котором изменяется связывание переменных	С точностью до расхода памяти
Emd	d) label L;	label <список_меток>;	модель из отдельного асоциативного списка для предстоящего связывания метки с позицией помеченного оператора	С точностью до расхода памяти

Eme
EMa
	e) L: a := 1;	goto 1;
		s_nov:= 0;
		step:= step*2;
		h:= h/2;
		1: int1 := -10;
			модель из отдельного асоциативного списка, связывающего метку с вхождением оператора	С точностью до скорости перехода

Eca ECa	a) (1*(2+X))	s_nov+f(a+(step-1)*h)	(EVAL NIL) (EVAL T) (EVAL (CONS T NIL)) (EVAL A)	С точностью до пространства возможных схем вычислений

Ecb
ECa
	b) S1 ; S2	s_star:= s_nov;
		s_nov:= 0;
		step:= step*2;
			(EVAL (CONS T NIL))
			(QUOTE A)
				С точностью до порядка вычислений

		h:= h/2;	(QUOTE (EVAL A)) (EVAL (QUOTE A))	
ECc ECb	c) ключевые слова	•**type** sym = (a, b, . . .); •**var** s: int; •**label** <список_меток>; •**goto** 1;	**NIL** (QUOTE A) (QUOTE (EVAL A)) **(EVAL (QUOTE A))** •вызовы функций	С точностью до «монады» = правила интерпретации
ECd ECb	d) goto L;	goto 1;	(QUOTE (EVAL A)) (EVAL (QUOTE A))	С точностью до откладывания вычислений
Esa ESa	a) s: array [. . . 9] of int;	var a1: array[char] of integer; - 256 компонент a2: array [char] of integer; - 256 целых компонент	(CONS NIL NIL) (CONS x y)	С точностью до размеров блоков памяти
ESb	b) s [5] := x	a2['z']:= 1; a3[-10]:= 2.5;	Допускают модель из асоциативного списка	С точностью до расхода памяти
Esc ESb ESc	c) x := s [5]	a4:= a2['z']+1;	(CAR (CONS x y)) (CDR (CONS x y))	С точностью до разницы между произвольным и последовательным доступом
	- для семантических систем базиса Лиспа существуют функционально подобные семантические системы базиса Паскаля с точностью до наполнения основных множеств алгебраических систем; - семантические системы базиса Паскаля **Eeb, Ema, Emb, Emd, Esb** характеризуют дистанцию между базисами, преодолимую с помощью моделей средствами базиса Лиспа, что можно рассматривать как расширенную семантику; - ряд обучающих фрагментов программ Паскаля можно разделить на две части: первая имеет эквиваленты на уровне базиса Лиспа, вторая - на уровне его расширений с помощью библиотечных функций, моделирующих средства Паскаля, отсутствующие в базисе Лиспа.			

Использование шкалы сопоставимых постановок задач и методов их решения, включая прагматику СП, даёт критерии для разработки подходов к оптимизации программ и систем программирования, уточнения методов измерения трудоёмкости программирования и производительности программ. Для каждого ЯП определены списки поддержанных в нём парадигм, предшественники, сфера влияния. Такие характеристики можно выводить из списков общих семантических систем, оценивая существенную разницу в традиционной прагматике функционально эквивалентных систем.

3. Структура измерительного стенда. Анализ определения языка программирования начинается с понимания задач, требований и поддержанных языком парадигм

программирования. В результате можно выделить понятийные таблицы, характеризующие монопарадигмальные подъязыки ЯП, работа с которыми сводится к следующим частям:

1) **Обработка понятийных таблиц**: выборки, вырезки, перестановки.
2) **Накопления результатов сравнения:** наследование и влияние.
3) **Наполнение обучающего ряда:** сопоставимые фрагменты программ.
4) **Тестирование и измерение производительности**: вывод зависимостей.

Таблица 7. Схема измерительного стенда ПРИЗМА

1. **Обработка понятийных таблиц** и генерация матриц для поддержки экспертных работ по сравнению разных ЯСП и форм для наполнения БД фрагментами отлаженных программ	2. **Накопления результатов сравнения** и экспертной оценки разных ЯСП, их хранение, лаконичное представление и вывод некоторых статистических и аналитических характеристик.
3. **Наполнение обучающего ряда** фрагментов отлаженных программ на разных ЯСП в форме, удобной для эксперимента и измерений, а также для формирования учебных пособий и справочников.	4. **Тестирование и измерение производительности** изучаемых ЯСП, организация хранения, статистического анализа и представления результатов измерений.

При организации стенда ПРИЗМА учтено, что большинство современных ЯП мультипарадигмальны. Поэтому их характеристика представляется комплексом таблиц. Для языка Си таких таблиц четыре (скалярная, процедурная, императивная, структурная), для более сложных может быть более десяти. Более тонкий анализ требует выборки-вырезки из таблицы клеток, строк, столбцов, миноров, элементов клеток, а также перестановок в зависимости от парадигмы. Для большинства ЯП известны их предшественники и сфера их влияния. Представляет интерес формализовать разницу с предшественниками и потомками, что может дать основания для создания лаконичных описаний ЯП, апеллирующих к ранее изученным. Практика изучения ЯСП базируется на экспериментах, для систематизации которых полезно выстраивать упорядоченные по сложности ряды фрагментов программ, приспособленных к прогону на реальных СП и пригодных для интуитивной и формализуемой оценки различий в семантике и прагматике реализации ЯП. Тестироование и измерение производительности ЯП в настоящее время выполняется как эксперимент над сложным комплексом из ЯП, СП, ОС и аппаратуры. Чтобы расслоить зависимость от таких компонентов, нужны канонические ряды материала, пригодного для обоснования роли программируемых решений в измеримых характеристиках программ. Общая схема системы ПРИЗМА представлена в таблице 7.

Заключение. Программирование давно стало массовой профессией, требующей объективных метрик для оценки качества программных изделий. Разработке таких метрик препятствует слишком высокий темп развития ИТ-индустрии, при котором доминируют быстрые интуитивные или волевые решения.

Предложен подход, использующий учёт особенностей решаемых задач и парадигм программирования, влияющих на выбор методов их решения, в зависимости от приоритетов в выборе языковых средств и реализационных конструкций. Намечена линия, позволяющая

сравнивать языки, выделяя сопоставимые примеры программ, и анализировать результаты прямых измерений, производительности программ, учитывая особенности базовых средств и реализационных решений в системах программирования.

СПИСОК ЛИТЕРАТУРЫ

1. Бентли Д. Жемчужины творчества программистов. Москва: Издательство «Радио и связь». Редакция переводной литературы. 1990. 217 с.
2. Васючкова Т.С. Методы измерения качества программного обеспечения. Препринт № 339, Новосибирск, ВЦ СО АН СССР. 1981. 30 с.
3. Городняя Л.В. Методика парадигмального анализа языков и систем программирования //Труды XXI Всероссийской научной конференции «Научный сервис в сети Интернет». М.: ИПМ им. М.В.Келдыша. 2019. С. 262-277. Режим доступа: http://keldysh.ru/abrau/2019/theses/03.pdfdoi:10.20948/abrau-2019-03
4. Городняя Л.В. О представлении результатов анализа языков и систем программирования. Труды XX Всероссийской научной конференции «Научный сервис в сети Интернет». М.: ИПМ им. М. В. Келдыша, 2018.
5. Городняя Л.В. Парадигмальная декомпозиция определения языка программирования // Труды XVIII Всероссийской научной конференции «Научный сервис в сети Интернет». М.: ИПМ им. М.В. Келдыша. 2016. С. 115-127. Режим доступа: http://keldysh.ru/abrau/2016/21.pdf
6. Городняя Л. В. Парадигма программирования. Учебное пособие. Спб: Лань. 2019. 232 с. ISBN 978-5-8114-3565-4
7. Городняя Л. В. Парадигмы программирования: анализ и сравнение. Новосибирск, СО РАН «Наука». 2017. 232 с. Режим доступа: https://www.rfbr.ru/rffi/ru/books/o_2043045
8. Колмогоров А.Н. Три подхода к определению понятия «количество информации». − Проблемы передачи информации. 1965. № 1 (1). С. 3-11.
9. Лаврищева Е.М. Программная инженерия и технологии программирования сложных систем. Учебник для вузов. М. 2018. 432 с.
10. Лавров С.С. Методы задания семантики языков программирования // Программирование. 1978. № 6. С. 3-10.
11. Фуксман А.Л. Технические аспекты создания программных систем. М.: Статистика. 1979. 180 с.
12. Peter Van Roy. Диаграмма с результатами сравнения более 30-ти парадигм программиро- вания. Режим доступа: https://www.info.ucl.ac.be/~pvr/paradigmsDIAGRAMeng108.pdf .
13. 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.
14. Диаграмма, представляющая хронологию появления и наследования многих ЯП. Режим доступа: https://www.levenez.com/lang/ –
15. Сайт с описаниями 171 языка и 31 парадигмы Режим доступа: http://progopedia.ru/ –.

UDK 004.43 (042.4)

THE PROJECT OF THE INFORMATION STAND "PRISMA" FOR THE PRESENTATION OF THE MEASURABLE CHARACTERISTICS OF PROGRAMMING LANGUAGES AND SYSTEMS

Lidia V. Gorodnyaya
Ph.D., Associate Professor, Senior Researcher
Ershov Institute of Informatics Systems (IIS)
Siberian Branch of the Russian Academy of Sciences
e-mail: gorod@iis.nsk.su
Sergey E. Demidov, Mikhail D. Kirichenko, Denis D. Tkachenko
Students of the Novosibirsk State University
e-mail: sersh_96@mail.ru, m.kirichenko@g.nsu.ru, **d.panfilov@g.nsu.ru**

Abstract. The article is devoted to issues arising in connection with the problem of measuring the characteristics of languages and programming systems that affect the complexity of software development and the productivity of software applications. Paradigmatic models of languages and programming systems can be useful in systematizing programming languages, assessing their similarities and differences, which allows us to build concise definitions regarding such models. This makes it possible to stratify the presentation of the features of the semantics of programming languages into autonomously developed components in the process of step-by-step development of experimental programming systems and the formation of schemes for studying and teaching system programming.

Keywords: systematization of programming languages, paradigm models, autonomously developed components, teaching schemes for system programming, concise definitions.

References

1. Bentley D. Zhemchuzhiny tvorchestva programmistov [The Pearls of creativity of pogrammers]. Moscow: Izdatel'stvo «Radio i svjaz'» = Publishing House "Radio and communication". 1990. 217 p. (in Russian)
2. Vasyuchkova T.S. Metody izmerenija kachestva programmnogo obespechenija [Methods for measuring software quality]. Preprint № 339, Novosibirsk, Computing Center of the Siberian Branch of the USSR Academy of Sciences. 1981. 30 p. (in Russian)
3. Gorodnaya L.V. Metodika paradigmal'nogo analiza jazykov i sistem programmirovanija = The techniques of paradigm analysis of languages and programming systems // Trudy XXI Vserossijskoj nauchnoj konferencii «Nauchnyj servis v seti Internet» = Proceedings of the XXI All-Russian Scientific Conference «Scientific service on the Internet». M : IPM named M.V. Keldysh. 2019 . Pp. 262-277. Available at: http://keldysh.ru/abrau/2019/theses/03.pdfdoi:10.20948/abrau-2019-03 (in Russian)

4. Gorodnyaya L. V. O predstavlenii rezul'tatov analiza jazykov i sistem programmirovanija. [On the presentation of the analysis results of the languages and programming systems.]. Trudy XX Vserossijskoj nauchnoj konferencii «Nauchnyj servis v seti Internet» = Proceedings of the XX All-Russian Scientific Conference «Scientific service on the Internet». Moscow: IPM named M. V. Keldysh, 2018. (in Russian)
5. Gorodnaya L.V. Paradigmal'naja dekompozicija opredelenija jazyka programmirovanija [Paradigmal decomposition of the definition of a programming language] // Trudy XVIII Vserossijskoj nauchnoj konferencii «Nauchnyj servis v seti Internet» = Proceedings of the XVIII All-Russian Scientific Conference «Scientific service on the Internet». Moscow: IPM named M. V. Keldysh,, 2016 . Pp. 115-127. Available at: http://keldysh.ru/abrau/2016/21.pdf (in Russian)
6. Gorodnaya L. V. Paradigma programmirovanija.[The programming paradigm]. Uchebnoe posobie = Tutorial. Sankt-Peterburg: Lan' = St. Petersburg: Doe. 2019. 232 pp. ISBN 978-5-8114-3565-4 (in Russian)
7. Gorodnyaya L.V. Paradigmy programmirovanija: analiz i sravnenie [Paradigms of programming: analysis and comparison] / Novosibirsk SO RAN: Nauka = Novosibirsk, SB RAS: “Science”. 2017. 232 p. Available at: https://www.rfbr.ru/rffi/ru/books/o_2043045 (in Russian)
8. Kolmogorov A.N. Tri podhoda k opredeleniju ponjatija «kolichestvo informacii» [Three approaches to the definition of "amount of information"]. Problemy peredachi informacii = Problems of information transfer. 1965. № 1 (1). Pp. 3–11. (in Russian)
9. Lavrishcheva E. M. Programmnaja inzhenerija i tehnologii programmirovanija slozhnyh sistem [Software engineering and programming technologies of complex systems]. Tutorial for high schools . Moscow. 2018 г. 432 p. (in Russian)
10. Lavrov S. S. Metody zadanija semantiki jazykov programmirovanija [Methods of definition the semantics of programming languages] Programmirovanie = Programming, 1978. № 6. Pp. 3-10. (in Russian)
11. Fuksman A.L. Tehnicheskie aspekty sozdanija programmnyh system [Technical aspects of creating software systems]. Moscow: Statistika = Statistics., 1979. 180 p. (in Russian)
12. Peter Van Roy. Diagramma s rezul'tatami sravnenija bolee 30-ti paradigm programmirovanija. [Chart with the results of comparison of more than 30 programming paradigms]. Available at: https://www.info.ucl.ac.be/~pvr/paradigmsDIAGRAMeng108.pdf
13. 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.
14. Chart representing the history of the emergence and inheritance of many programming languages. Available at: https://www.levenez.com/lang/
15. Website with descriptions 171 languages and 31 paradigms. Available at: http://progopedia.ru/