УДК 004.43 (042.4)

ОТ ТРУДНО РЕШАЕМЫХ ПРОБЛЕМ К ПАРАДИГМАМ ПРОГРАММИРОВАНИЯ

Городняя Лидия Васильевна
к.ф.-м.н., доцент, с.н.с., e-mail: gorod@iis.nsk.su,
Институт систем информатики СО РАН,
630090 г. Новосибирск, проспект Академика Лаврентьева, 6.

Аннотация. Статья посвящена описанию и анализу взаимосвязи между трудно решаемыми проблемами и процессом формирования парадигм программирования, которые могут быть полезны при решении таких проблем. Материал для анализа даёт история вычислительной техники, а также малозаметная линия научной деятельности лидера программирования А.П. Ершова в области проявления и развития парадигм программирования[1]. Результаты анализа позволили выполнить систематизацию языков программирования, представляя оценку их сходства и различия, что позволяет строить лаконичные определения относительно парадигмальных моделей. Это даёт возможность стратификации особенностей семантики языков программирования с разделением на автономно развиваемые компоненты в процессе пошаговой разработки экспериментальных систем программирования и формирования схем изучения и преподавания системного программирования, что может быть полезным для повышения производительности программного обеспечения. Попутно формируется ранжирование постановок задач по степени их изученности, что влияет на оценку трудоёмкости программирования.

Ключевые слова: систематизации языков программирования, парадигмальные модели, лаконичные определения, А.П. Ершов, парадигма программирования, стратегические парадигмы, проблемы программирования, степень изученности задач, архитектура фон Неймана.

Цитирование: Городняя Л. В. От трудно решаемых проблем к парадигмам программирования //Информационные и математические технологии в науке и управлении. 2021. № 1 (21). С. 94-109. DOI:10.38028/ESI.2021.21.1.008

Введение. Многие проблемы современного программирования связаны с недостаточным учётом гуманитарных аспектов любой человеческой деятельности [1]. Известно, что успехи пионерных проектов Новосибирской школы программирования, выполненных в сфере влияния академика А.П. Ершова в ВЦ СО АН СССР, достигнуты в условиях исключительного внимания человеческому фактору в вопросах подбора кадров и создания в Новосибирском Академгородке плодотворной обстановки для выполнения всех необходимых целевых и вспомогательных работ по применению вычислительной техники. Широкий контекст и концентрация исследовательской и конструкторской активности позволили именно в Новосибирском Академгородке сформулировать перспективы исследования и развития системной информатики, включая ключевое значение образовательной информатики как механизма профориентации в программировании. Тем не менее, до сих пор не получила четкой формулировки общая характеристика предмета информатики как фундаментальной науки и системного программирования как самостоятельной деятельности. За исключением вопросов информационной безопасности, признание стратегического значения исследований в области ИТ не привело к поддержке актуальных направлений развития системной информатики, являющейся несущей линией развития устойчивых, надёжных и безопасных информационных систем и технологий. Не описаны механизмы поиска и реализации рациональных решений трудных проблем, обременённых сложно удостоверяемыми критериями качества и ошибками в прогнозе трудоёмкости их решения.

В сфере программирования наиболее яркой является школа академика Андрея Петровича Ершова. Научная карьера Андрея Петровича затрагивает четыре направления:

[1]Термин «парадигмы программирования» предложил Роберт Флойд в 1978 году в своей Тьюринговской лекции [12] .

программные эксперименты, теория программирования, общественная деятельность по формированию программистского сообщества, постановка школьной учебной дисциплины «информатика». В науке лидерство означает умение распознавать проблемы и расширять пространство путей их решения. А.П. Ершов оказался способным распознавать неожиданные ключевые проблемы программирования как новой области и убеждённо расширять пространства решений таких проблем. Коллеги и ученики Андрея Петровича освещают преимущественно линию построения компиляторов, практическая значимость которой наиболее очевидна. Остальные направления служат фоном, их признание и проявление требуют более тонких исследований, результат которых могут не дождаться подтверждения очевидцами и участниками. При систематизации парадигм программирования[2] бросается в глаза, что многие системообразующие и перспективно-стратегические парадигмы программирования не просто входили в сферу интересов Андрея Петровича, но в формирование многих из них ему удалось сделать значительный вклад, другие − получили развитие трудами его учеников [2]. Отдельно следует отметить концепцию лексикона программирования [3].

Статья начинается с общего представления о парадигмах программирования [4]. и краткого изложения подхода к описанию этого термина в Тьюринговской лекции Роберта Флойда [5]. Далее выполнено сравнение близких понятий с анализом их специфики, сводимой к проверяемым правилам, показывающим роль понятий в процессе решения задач с учётом степени их изученности [6]. Затем показана зависимость появления и признания парадигм от их практичности и изученности решаемых задач, а также связанность лексикона программирования с параллельно развивающимися стратегическими исследованиями на уровне теории [7, 8]. Таким образом, выстраивается последовательность расширения класса практических задач, проявления в нём трудно решаемых проблем и формирования парадигм как общих интуитивных схем для снижения трудоёмкости представления и отладки программ решения трудных проблем [9]. В результате можно классифицировать пространство постановок задач и формулировать требования к современным языкам и системам программирования (ЯиСП) для решения новых трудно решаемых проблем, возникающих в связи с расширением возможностей аппаратуры, ростом квалификации пользователей и разработчиков, а также общим прогрессом ИТ-индустрии [10, 11, 12].

1. Общее представление. Словарное определение термина «парадигма» сводит его к понятиям «пример, модель, образец». Такого понимания не достаточно для ответов на практические вопросы программирования:

- Почему стремительно растёт число ЯиСП?
- Что позволяет обосновывать выбор ЯиСП для решения конкретной задачи?
- Какие параметры позволяют определять, какую парадигму поддерживает ЯиСП?
- Какова трудоёмкость разработки программы на базе определённых ЯиСП?
- Что отличает Парадигму программирования от технологии, методики, стиля, техники, метода, вида и др. близких или сопутствующих понятий?
- Кто отвечает за правильность и производительность программ?

Мировая практика насчитывает несколько десятков тысяч языков программирования (ЯП) [13].Долгоживущие, учебные и новые ЯиСП преимущественно мультипарадигмальны [14]. Понимание «правильности» по разному рассматривается в теории и практике. Прогноз трудоёмкости программирования обычно содержит систематическую ошибку в 2-8 раз. Измерение производительности готовых программ не даёт основания для вывода

[2] Достаточно конструктивное определение понятию «парадигмы программирования» дал Питер Вегнер, связывая его с условиями, которые могут быть проверены [5]. Р. Флойд ввёл этот термин без формального определения, просто показал на примерах.

направлений их улучшения и вклада программируемых решений. Расхождение интуитивных и измеримых характеристик сложных систем (ошибка конъюнкции) выдерживает научно-образовательную коррекцию даже среди математиков.

2. История термина. Вернёмся в 1978 год, когда Роберт Флойд провозгласил: «Искусство программирования включает в себя расширение репертуара используемых парадигм» [5]. Этой проблеме он посвятил свою Тьюринговскую лекцию с акцентом на проблему обучения программистов, влияния парадигм программирования на успех программистских проектов и как парадигмы должны быть поддержаны в языках программирования. Прежде всего он счёл нужным отметить пробелы в программистском образовании, по его мнению, потеряны рекурсивные сопрограммы, бывшие в языке Fortran, слабо осваиваются недетерминизм и макротехника, а также концепции языка Lisp, активно использующие программы как данные в соответствии с принципами архитектуры фон Неймана. Выход из всего этого Р. Флойд видел в переходе к обучению семантическим методам, позволяющим без чрезмерного синтаксического разнообразия продвигаться от упрощённой учебной постановки задачи до непрерывно расширяющегося класса практических задач.

Не приводя определение термина, Роберт Флойд в качестве примера привёл «Структурное программирование» как методологически доминирующую парадигму, нацеленную на нисходящее проектирование, пошаговое улучшение, сведение задачи к более простым подзадачам и переход от конкретных данных и процедур машинного уровня к более абстрактным объектам и функциям, позволяющим продумывать модули, выделяемые при нисходящем проектировании.

Трудности учебного процесса в отмеченных Флойдом проблемах не случайны. Рекурсивные сопрограммы не изучаются в учебной практике − их трудно отлаживать, не было удобных параллельных систем программирования для лабораторных работ. Недетерминизм изучается как теория без перехода к производству, т. к. хватает более изученных задач, с ними работать легче. Макротехнику трудно отлаживать и проверять результат. Идея языка Lisp, что программы можно обрабатывать, как данные, противоречит традиции императивно-процедурного программирования и требует перехода к сложному абстрагированию. Обучению семантическим методам препятствует математический ценз, необходимый для перехода от упрощённой учебной постановки задачи к более сложным и абстрактным. Упрощённые кажутся несерьёзными и для производства бесполезными, а возникающие потом непрерывно расширяющиеся и альтернативные классы практических задач влекут страх перед новыми проблемами, решения которых преподавателю не известны.

Понимание любой парадигмы скрывает переход от имплицитного, интуитивного образа мышления и скрытой грамматики решения проблем к системе осознанных понятий и приоритетов, используемых в определённом конкретном виде деятельности или поведения, включая речевое поведение или вербализацию решений, а для программирования − выбор схемы жизненного цикла программ их решения и стиля программирования, влияющих на трудоёмкость отладки программ и их жизнеспособность. Наследуя историю понимания и восприятия, следует отметить, что парадигмы имеют специфику, связанную с предпочтением программируемого эксперимента, похожего на удостоверение результатов в физике.

Следует отметить, что списки парадигм в описаниях языков программирования (ЯП) могут не соответствовать смыслу парадигм [15]. Обычно известны предшественники и сфера влияния ЯП. Заметно отставание осознания парадигм от их практического

применения на 10-15 лет. Появляется инструмент, потом вдруг обнаруживается, что он поддерживает новую парадигму.

3. Сопутствующие и близкие понятия. Отдельный вопрос лексикона программирования − отличие парадигм от других сопутствующих и близких понятий, слабо отличающихся в речевой практике [14, 16]:

- Технология − комплекс методик для гарантированного получения результатов программирования при выполнении определённых условий, соответствующих постановке задачи.
- Методика − готовый рецепт для проведения каких-либо целенаправленных действий.
- Стиль − форма деятельности, характеризующая особенности общения, манера себя вести, методы и приёмы работы, возможно, отражающие склад мышления.
- Техника − устройства для повышения эффективности труда, а также снижения вероятности ошибок человека при выполнении каких-либо сложных операций.
- Метод − способ достижения цели, совокупность приемов и операций практического и теоретического освоения действительности.
- Формы − типичный образец оформления программ.
- Виды − характеристика составляющих программы, их внешнее представление.

Технология может использовать комплекс парадигм, вызванных разнообразием требований к отдельным фазам производства. Методика − может быть частью парадигмы без учёта остальных аспектов. Стиль − может внешне быть похожим, но содержательно не соответствовать решаемой задаче. Техника − может быть частью парадигмы без принципиального влияния на успех в программировании решения задач. Метод − может быть частью парадигмы на ограниченном классе задач, а на других выходить за пределы её успешного применения. Формы − могут внешне быть похожими, но без содержательного соответствия решаемым задачам. Виды − могут маскировать иные подходы, даже противоречащие провозглашаемой парадигме.

Соглашаясь, что парадигма − это образец, рассмотрим несколько часто цитируемых определений[3].

Диомидис Спинеллис**:** «Слово «парадигма» используется в программировании для определения семейства **обозначений** (нотаций), разделяющих общий способ (методику) реализаций программ» − реально это сведение к синтаксису.

Венская методика в конце 1950-ых констатировала, что сводимые к общему абстрактному синтаксису ЯП эквивалентны. Синтаксис – предельно изученный в теории аспект представления программ, определены нормализованные и канонические формы, достаточные для автоматического конструирования анализаторов и генераторов программ. Практика обучения программированию показывает, что на синтаксис можно не обращать внимания, т. к. ошибки такого рода ловит система программирования, что побуждает быстро научаться их исключать.

Многие определения сводят термин «парадигма» к понятию «подход»:

- Дэниел Бобров (D. G. Bobrow): парадигма определяется как «стиль программирования как описания **намерений** программиста».
- Брюс Шрайвер (Bruce Shriver): определяет парадигму программирования как «**модель** или подход к решению проблемы».
- Линда Фридман (Linda Friedman): «подход к решению проблем программирования».

[3] Определения из раздела «Парадигма программирования» Википедии

- Памела Зейв (Pamela Zave): «способ размышления о компьютерных системах» (в оригинале «way of thinking about computer systems»).

Такие определения отражают уровень «технического задания», но не поясняют причин развития репертуара парадигм и расширение спектра задач программирования. Несколько детальнее определение парадигм как способа или метода.

Тимоти Бадд (T. A. Budd.) : «способ концептуализации того, что значит „производить вычисления“, и как задачи, подлежащие решению на компьютере, должны быть структурированы и организованы». - Это уровень «Технического проекта», но нет перехода собственно к программированию после проектирования, а также к отладке и развитию программ.

Имеется и определение парадигмы как классификации и проверки выбора нужных средств: Питер Вегнер (Peter Wegner)[4] предлагает в его работе «Concepts and paradigms of object-oriented programming» парадигмы определять как «правила классификации языков программирования в соответствии с некоторыми условиями, которые могут быть проверены» [16].

4. Проверяемые правила и новизна задач. Возникает вопрос: какие параметры позволяют различать парадигмы программирования и сферу их успешного применения? Насколько возможно измерение зависимости успеха программирования от выбора парадигмы (трудоёмкость, производительность)? Логично предположить, что измерение зависимости успеха программирования связано с оценкой степени изученности решаемых задач [17]. Это можно проиллюстрировать на аналогии с решением проблем создания компьютерного мира.

Макетный образец – концептуальный минимум, показывающий произошедшее **открытие** новых задач, ранее их не замечали, абсолютно нет образцов для сравнения пользы от их решения (аналитическая машина Ч. Беббиджа).

Экспериментальный полигон – потенциальный максимум, **изобретение** альтернативных и исследование как можно более полного пространства пробных решений (авторы первых компьютеров – «хранимая программа»)

Практичная версия – производственный компромисс, **рационализация** уже известных готовых решений обычных, типовых постановок задач и их улучшение ради экономической целесообразности снижением трудозатрат (языки и системы программирования, базы данных, операционные системы, программные инструменты).

Эффективная реализация – **стабилизация** предельного баланса трудоёмкости и производительности для решения точно поставленных задач, улучшение которого не оправдано на текущем уровне знаний (требования и качество, «правильность», эталоны, готовые встраиваемые модули).

История программирования для компьютеров начиналась с решения точно поставленных задач, достаточно изученных в докомпьютерную эпоху и потому приспособленных к нисходящей методике программирования. Решения новых задач формируются по восходящей методике, главное, по меткому замечанию А.Н. Томилина, чтобы была решена полезная задача [18]. Отмертм сразу, что компьютерная инженерия развивается на сравнительно стабильном классе задач усовершенствования техники. Программное производство поддерживает непрерывное расширение класса решаемых задач, связанных с кругом пользователей, элементной базой и стремительным развитием ИТ. Это объясняет резкий дисбаланс между прогрессом аппаратуры и отставанием понимания её возможностей пользователями, а также определяет вхождение в почти любой

[4] Вегнер П. Программирование на языке АДА. М. Мир. Пер. с англ. 1983. 240 с.

программистский проект некоторого числа новых задач, решение которых никому не известно, и ставит образовательную проблему самообучения программистов [19].

Открытия Беббиджа в начале 1830-х ждали признания почти сто лет, изобретения К. Цузе, В. Атанасова, Г. Айкена, Дж.П. Эккерта и др. в первой половине двадцатого века происходят в соревновании за приоритеты и патенты на изобретения, во второй половине начинается период рационализации индустрии элементной базы и программного обеспечения, теперь можно видеть стремление гарантировать надёжность и безопасность использования ИТ стабилизацией используемых средств [17].

Наследники Ч. Бэббиджа воспроизвели по его чертежам макетный образец аналитической машины на уровне технологической точности его времени, чем опровергли объяснение неудачи чрезмерным опережением технических возможностей. Основные трудности были связаны с общим непониманием проблемы автоматизации вычислений и отсутствием речевой практики обсуждения возможных решений. Для каждого открываемого элемента проекта приходилось придумывать названия, позволяющие приблизительно воспринимать и обсуждать его назначение. А. Тьюринг, Г. Айкен, В. Атанасов и Дж. фон Нейман упоминают о предварительном знакомстве с идеями Ч. Беббиджа и проектом аналитической машины, что позволило от открытий перейти к изобретениям на фоне уже понятного проекта макетного образца, лексикона, формальных моделей вычислимости и множества примеров отдельных ранее созданных вычислительных устройств. На этом же фоне К. Цузе и Дж.П. Эккерт изобретают свои машины независимо и самостоятельно. Здесь стоит принять во внимание существование проблем, имеющих единственное разумное решение, к которому с необходимостью приходит каждый конструктор без всякого заимствования. В этом кругу неслучайных решений Дж. фон Нейман приходит к идее иерархии памяти, согласно которой внешние устройства и средства ввода-вывода данных обмениваются с оперативной памятью, а не с арифметическим и логическим устройствами, как это было в более ранних архитектурах [20][5].

«2.7 Четвертое: устройство должно иметь органы для передачи (числовой или другой) информации от R в его определенные части, C и M. Эти органы образуют его вход, четвертую конкретную часть: I. Видно, что **это лучше всего делать все переводы из R (посредством I) в M, а не напрямую в C** (ср. f14.1, 15.3g).

2.8 Пятое: устройство должно иметь органы для передачи (предположительно, только числовой информации) из его конкретных частей C и M в R. Эти органы образуют его выход, пятую конкретную часть: O. Будет видно, что **снова лучше делать все переводы из M (посредством O) в R, и никогда напрямую из C** (см. f14.1, 15.3g) ».

Если степеням изученности решаемых проблем сопоставить последовательность парадигм программирования: логическое, функциональное, объектно-ориентированное и императивно-процедурное, то можно сделать вывод, что парадигма программирования – это проявление образа мышления в принятии решений, зависящего от степени изученности постановок задач, выбора схемы жизненного цикла программ и стиля программирования, влияющих на трудоёмкость отладки программ и их жизнеспособность. Принятие решений с учётом прогноза развития может учитывать и другие параметры:

- Легко – зависимость от одного параметра.

[5] «2.7 Fourth: The device must have organs to transfer (numerical or other) information from R into its speci c parts, C and M. These organs form its input, the fourth speci c part: I. It will be seen that it is best to make all transfers **from R (by I) into M, and never directly into C** (cf. f14.1, 15.3g).

2.8 Fifth: The device must have organs to transfer (presumably only numerical information) from its speci c parts C and M into R. These organs form its output, the fth speci c part: O. It will be seen that it is again best to make all transfers **from M (by O) into R, and never directly from C**, (cf. f14.1, 15.3g).» [20]

- Часто – взаимодействие двух независимо изменяющихся параметров.
- Редко и трудно – взаимодействие большего числа параметров.

Для мультипарадигмальных ЯП отдельная парадигма – аналог проективной геометрии или плоских проекций в инженерии, дифференциального анализа в математике.

Инженеров учат строить проекции, по которым можно вообразить и сделать объёмную конструкцию. Проверка принадлежности или поддержки парадигмы подразумевает возможность измеримых характеристик (трудоёмкость, производительность), допускающих внешнюю демонстрацию [10, 11, 21].

5. Изученность и абстрагирование. В практике парадигма программирования (ПП) является технологичным инструментом интуитивного грамматического описания фактов, событий, явлений и процессов, возможно, не существующих одновременно, но объединяемых в общее понятие в рамках деятельности по созданию свободных от ошибок решений нужных задач.

Различие основных ПП подобно разнице в уровне изученности решаемых задач:

- предельная эффективность (готовый алгоритм),
- практичная версия (зрелый лексикон приложения),
- экспериментальный максимум (исследование),
- концептуальный минимум (не вполне ясная перспектива).

Выделяются категории (монады) языковых средств, подобные частям процессора и конфигурациям аппаратуры, а вместе с ними – новые проблемно-ориентированные языки, лишь отчасти решающие проблемы понятного и лаконичного описания семантики языков и систем программирования, их реализационной и эксплуатационной прагматики, выделения из программ автономно развиваемых компонентов, учёт механизмов имплицитного обучения, использования интуиции и неявного знания [9,19].Функциональное программирование обладает высокой моделирующей силой, его можно использовать как нормализующую форму при сравнении семантики [6].

Существует качественное различие между решением **простых** задач на небольшом числе примеров, представимых легко обозримыми данными, и решением **сложных** задач, требующих анализа большого разнообразия редко встречающихся ситуаций и трудно обозримых данных, осложнённых повышенными требованиями к правильности, надёжности и другим критериям. Нет признания, что устанавливается соответствие осознанных и интуитивных представлений, в частности, выражающееся разницей уровней абстрагирования понятий задач и языков программирования, преодоление которой требует представления схем и отдельно – фрагментов для их наполнения [1].

Независимо от парадигмы программирования трудоёмкость тесно связана с уровнем абстрагирования используемых понятий, часть которых могут не иметь аналогов в речевой практике:

- Низкоуровневое программирование – возможность реализации эффективных решений ценой использования общего доступа к любым слабо защищенным структурам данных.
- Языки высокого уровня – использование иерархии, независимые компоненты которой защищены от бесконтрольного взаимодействия.
- Средства сверх высокого уровня (языки спецификаций, языки параллельного программирования, системы представления знаний и т.д.) – полнота пространства решений по отдельным направлениям проблем, связанных с разработкой и применением программ.
- Проблемно-ориентированные языки (DSL) – переход от наследования библиотек к наследованию подъязыков.

6. Практичные и стратегические парадигмы программирования. Внешне различие парадигм программирования (ПП) сводится к приоритетам в оценке важности понятий, используемых для решения задач разработки программ или изучения методов программирования в четырёх направлениях: вычисления, управление вычислениями, хранение данных и их укрупнение – комплексы. Известно, что усложнение по одному направлению обычно не вызывает затруднений, два направления уже требуют некоторых усилий, три или четыре направления часто оказываются принципиальным препятствием для понимания. Это позволяет дать формальное описание типовых категорий семантических систем, характерных для поддержки конкретных базовых ПП и оценки их сложности. Описанию производных ПП в таком случае можно сопоставлять нечто вроде семантического индекса, отражающего разницу с базовой ПП. К базовым парадигмам в настоящее время принято относить императивно-процедурное (ИПП), функциональное (ФП), логическое (ЛП) и объектно-ориентированное (ООП) программирование, а также иногда параллельные вычисления (ПВ). Альтернативой, преимущественно в образовательной среде, является выделение в качестве фундаментальных парадигм функционального, параллельного и императивно-процедурного программирования, а логическое и объектно-ориентированное относят к дополнительным парадигмам, преподавание которых обременено необходимостью знания некоторых предметных областей.

Постановки задач в рамках ИПП начинаются с определённого алгоритма решения задачи, для которого прежде всего принимают решения по организации хранимых данных и логике управления их обработкой (информационно-логический граф). Производные ИПП выделяют разные методы представления данных и организации порождаемых программой последовательных процессов: автоматное – без процедур; скалярное – без структур данных; структурное – регулярная логика и типы данных; сборочное – крупноблочность; диаграммно-графовое – визуальность; сценарное – одноуровневое, поверхностное; операционное – управление заданиями; языки действий – управление акторами; векторно-ориенированные – обработка массивов; синхронизация над общей памятью и т. п.

Постановка задач ООП основана на доступе к иерархии классов объектов с готовыми работоспособными методами решения ряда задач некоторой сферы приложения ИТ. Требуется без лишних трудозатрат уточнить эту иерархию, чтобы приспособить её к решению новых востребованных задач этой области, её расширения или ей подобной. Производные ООП дают разнообразные конкретизации понятия «класс объектов»: аспектно-ориентированное –разделение слоёв (вертикальное слоение А.Л. Фуксмана [21]); прототипная – без классов; табличная – с таблицами в роли классов; обобщённая – с абстрактными типами данных в роли классов; DSL – языково-ориентированная настройка над gcc; субъектно-ориентированное – объекты с поведением; агентно-ориентированное – выделение исполнителей; сервис-ориентированное – выделение обслуживания и т.п.

Задачи ФП ориентированы на известную предметную область, в рамках которой следует выбрать и отладить систему универсальных функций вычисления результатов по заданным аргументам, используя символьное представление данных с иерархией структур над ними, пригодное для создания и исследования прототипов решения задач из этой области. Производные ФП представляют собой вариации в методах организации вычислений: ленивые вычисления – откладывание действий; комбинаторное – сведение к функциям; гомоиконное – подобия структур данных и обрабатывающих программы; апликативное – подгрузка любого базиса; потоковый параллелизм – множество результатов; правило переписывания – подстановка; бесточечное – форма записи; алгебраическое – выделение семантических систем; инсерционное – динамические вставки-замены (точки

роста С.С. Лаврова [22]; грамматико – ориентированная обработка (BNF – формы Бэкуса-Наура); стек-ориентированное (Forth) и пр.

Решения задач ЛП исходят из заданной коллекции фактов, прецедентов и отношений, частично характеризующей новую задачу, пока не имеющую эффективного или практичного решения. Надо привести эту коллекцию к форме, демонстрирующей возможность ответов на практичные запросы относительно данной задачи. Производные ЛП используют разные подходы к смягчению зависимости получения результатов от случайного, избыточного или недостаточного детерминизма при переборе вариантов: откаты-бэктрекинг – перебор вариантов; программирование в ограничениях – границы определённости; представление знаний – привлечение экспертов; вопрос-ответное – задачи поиска; индуктивное – вывод решения; естественно-языковое – запросы на известном подмножестве естественного языка; онтологическое – системы понятий в предметной области; недетерминированное – без преждевременного упорядочения и др.

Постановки задач организации **параллельных вычислений** 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 от готовых решений к масштабируемым решениям сверхвысокого уровня, допускающим автоматическую настройку на реальные конфигурации оборудования, а также из-за отсутствия традиции представлять течение времени в классических математических моделях. Высокопроизводительные вычисления 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. Это позволяет рассматривать такие задачи, как бесспорный полигон для признания потенциала доказательных методов программирования и верификации. Есть основания доказательные методы рассматривать, как важную составляющую неограниченных парадигм, работоспособность которых зависит от качества автоматизированной проверки **правильности** функционирования программ.

Ряд **перспективно-стратегических** парадигм, ориентированных на совершенствование программирования как деятельности, получили особую поддержку академика А.П. Ершова. Перспективно-стратегические парадигмы программирования нацелены на повышение квалификации программистского корпуса, выделяя в качестве фундаментальных теоретическое, системообразующие, учебно-ознакомительное, экспериментальное и другие парадигмы программирования, отличия которых не сводятся к приаппаратной или реализационной характеристике, но могут быть выражены в терминах приоритетов и наполнения категорий семантических систем [17].

Парадигмы теоретического программирования базируются на схематологии, алгоритмическом, доказательном и верификационном программировании. Системообразующие парадигмы программирования, кроме процессорного уровня, учитывают особенности многократного использования внешней памяти и периферийных устройств, необходимых при разработке инструментария для накопительных эффектов обработки данных, включая системы программирования, операционные системы, базы данных и многопроцессорные комплексы. Здесь системное программирование отличается от поддержки отдельного прогона программы переходом к обеспечению серии прогонов и отладки улучшаемых программ, а также, операционных систем, систем управления базами данных и средств доступа к многопроцессорным комплексам. Отладка может рассматриваться как вторичное проектирование программы или повышение степени

изученности решаемой задачи – взаимная учёба разработчиков и пользователей. Учебное программирование дополняется олимпиадным, а также непрофессиональным и экспериментальным, позволяющим любителям программирования формировать свою квалификацию методом самообучения (табл. 1).

Таблица 1. Систематизация парадигм программирования.

Практичные	Парадигмы программирования	Стратегические	Парадигмы программирования
Базовые	Императивно-процедурное Функциональное Логическое Объектно-ориентированное	Фундаментальные	Алгоритмическое Теоретическое (схематология) Доказательное Верификационное
Технологичные	Оптимизационное Трансформационое (проекции) Отладочное и тестирование Свободно-распространяемое	Системно-расширяющие	Системное (компиляторы) Операционные системы Системы управления БД Многопроцессорные комплексы
Безграничные	Большеобъёмные данные Дистанционный доступ Параллельные вычисления Суперкомпьютерное (высокопроизводительное)	Образовательные	Учебное Олимпиадное Непрофессиональное Экспериментальное (любительское)

7. Трудно решаемые проблемы и парадигмы программирования. Часто утверждают, что основная задача программиста – это кодирование созданных математиками алгоритмов, выполнение подготовленных менеджерами технических заданий или реализация разработанных аналитиками спецификаций. Перспективно-стратегические парадигмы программирования академика А.П. Ершова различаются по обнаруживаемым проблемам и подходам к их решению, открывающим новые пространства решений, по мере изучения которых проблемы преобразуются в практические задачи с примерами типовых решений. Нахождение практичных методов и создание удобных средств приводит к исчезновению проблемы, кажется, что её и не было. Череда пройденных Андреем Петровичем парадигм программирования доказывает, что истинная **миссия программирования** – решение новых задач при отсутствии готовых средств и типовых методов, что делает актуальным самообучение программистов методам решения изобретательских и новых задач на любом материале [19]. Многие из этих парадигм для его коллег и учеников стали полигоном научных исследований, другие парадигмы обрели аналоги по мере развития ИТ, некоторые временно оттеснились от массового программирования, ждут своего времени [17]. Темп проявления новых проблем и появления парадигм программирования усложняет процесс формирования профессионального лексикона. Развитие современного программирования проходит фазу формирования парадигм параллельного программирования, связанных с проблемами обработки больших массивов данных, организации суперкомпьютерных вычислений и эксплуатации сетевых компьютерно – коммуникационных взаимодействий (табл. 2).

Таблица 2. Взаимосвязь между обнаружением проблем и проявлением парадигм.

№[6]	Проблемы	Практика	Перспективы
1	Практичность	Технологичное программирование	
2	Надёжность		Теоретическое программирование
3	Трудоёмкость	Системное программирование	
4	Результативность		Алгоритмическое программирование
5	Эффективность	Оптимизационное программирование	
6	Эквивалентность		Доказательное программирование
7	стартовый барьер	Ознакомительное программирование	
8	Обучаемость		Учебное программирование
9	Производство	Операционные системы	
10	Исследования		Мультипарадигмальное программирование
11	Новые машины	Многопроессорные комплексы	
12	Новые языки		Многоязыковое программирование
13	Интеллектуальные инструменты	Экспериментальное программирование	
14	Интеллектуальные технологии		Исследовательское программирование
15	Искусственный интеллект	Непрофессиональное программирование	
16	Пространство решений		Параллельное программирование
17	Свойства исполнимого кода	Типизация данных	
18	Правильность программ		Верификационное программирование
19	Доступ к новым источникам	Системы управления БД	
20	Лексикон		Популяризация знаний
21	Устойчивость систем	Конструктивное программирование	
22	Корректность программ		Трансформационое программирование
23	Полиграфия	Издательское программирование	

[6] Приблизительная последовательность обнаружения трудно решаемых проблем

24	Фактография		Редакционное программирование
25	Прогресс элементной базы	Школьная информатика	

8. Пространство постановок задач. По трудоёмкости решения вполне очевидно предпочитаются стабильные задачи, удобно использующие императивное программирование с добавлением обработки векторов. Несколько труднее решаются развивающиеся задачи, допускающие множественность решений, варианты и пользовательские классы понятий, обычно решаемые в рамках ООП, функционального или логического программирования. Возникает разнообразие структур данных, контроль типов данных и сигнатур операций, методов непрерывной разработки, отладки, тестирования и верификации программ.

Более сложные задачи требуют усложнения методов автоматизации программируемых решений, что с одной стороны дают языки сверх высокого уровня, а с другой стороны – создание новых языков программирования – DSL-языки. Значительная доля таких задач связана с необходимостью эффективной организации параллельных вычислений и процессов на современной технике, включая асинхронное программирование, параллельное программирование над общей памятью и синхронизацию функционирования независимо создаваемых и улучшаемых программ. Многие такие задачи обладают повышенными требованиями к надёжности, безопасности, жизнеспособности, производительности.

Разработка систем программирования для решения сложных задач обычно представляет собой обработку данных не менее четырёх форматов: текст, структуры, коды, библиотеки. Кроме проблем реализации обработчиков текстов на языках программирования, решаются менее формализованные, но не менее сложные задачи, связанные с методиками, инструментами и оборудованием, что существенно влияет на требования к уровню квалификации системного программиста.

9. Современные требования к языкам и системам программирования. Одна из болезненных проблем применения ИТ – необходимость их непрерывного улучшения, лишь отчасти связанного с обновлением аппаратуры. Техника улучшения реализуется как обновление операционных и прикладных программных систем, нередко сопровождаемое искажением функционирования сторонних инструментов. Не исключено, что такое положение дел навязано исходным убеждением, что «полезная» программа может быть создана раз и навсегда. Само понятие пользы подвержено незаметному развитию, влекущему необходимость пересмотра ранее выполненных решений, что приводит к итеративности процесса разработки программ решения практичных задач.

Другая проблема связана с пропускной способностью восприятия, для учёта возможностей которой нужна методика создания лаконичных и конструктивных форм, позволяющих сложные программы верифицировать по частям, анализируя их отдельные свойства без разрушения взаимосвязей в полной программе. Имеется и обратная проблема – расширяемость программ по мере развития класса частично решённых задач. Эта проблема ярко проявляется в задачах организации параллельных вычислений на фоне имеющихся программ решения тех же задач в обычной императивно-процедурной парадигме программирования.

Заключение. Таким образом, анализ и сравнение языков и систем программирования, а также их выбор при решении практических задач можно выполнять на основе классификации особенностей класса трудно решаемых проблем, поиска подходящих парадигм программирования, а при отсутствии парадигм с подходящим прогнозом трудоёмкости – изобретение новых парадигм программирования. Как показывает опыт

развития Новосибирской школы А.П. Ершова, такой путь может дать конструктивные результаты и обеспечить повышение уровня квалификации программистского корпуса, являющегося ключевым для достижения надёжности и безопасности современных ИТ [17].

Понимание разницы между парадигмами программирования как различий в упорядочении приоритетов принятия решений в процессе полного жизненного цикла программ, начинающегося с постановки задачи, позволяет наметить ответы на вопросы, перечисленные в начале статьи:

- Стремительно растёт число языков программирования вслед за столь же стремительным расширением пространства решаемых задач, часть которых ранее не были заметны.
- Для обосновывания выбора парадигм и языков программирования для решения конкретной задачи, следует определить степень её изученности и последовательность приоритетов, вытекающих из требований к её решению в виде программы.
- Параметры, позволяющие определять, какие именно парадигмы поддерживает язык программирования, проявляются при выделении монопарадигмальных подъязыков, схема понятий которых удобно упорядочивается в соответствии с предполагаемыми парадигмами.
- Прогнозировать трудоёмкость разработки программы на базе определённой системы программирования можно по оценке степени изученности решаемых подзадач и поддержке системой программирования выбираемых парадигм.
- Парадигму от технологии, методики, стиля, техники, метода и других понятий отличает её обусловленность скрытой, интуитивной грамматикой деятельности, образом мышления, пониманием приоритетов для успешного полного жизненного цикла программ, а не только собственно программирования.
- Программист отвечает за правильность и производительность программ даже в тех случаях, когда ему навязывают выбор средств и методов потому, что не устранима возможность дополнять систему программирования библиотечными модулями, способными компенсировать исходное отсутствие нужной парадигмы.

Всё вышеизложенное можно рассматривать как методику формирования лексикона программирования в современных условиях развития ИТ.

СПИСОК ЛИТЕРАТУРЫ

1 Городняя Л.В. Гуманитарные факторы программирования. Новосибирск: СО РАН. 2020 г. 163 с.. ISBN 978-5-6044349-4-9

2 Городняя Л.В. Перспективно стратегические парадигмы программирования Академика Андрея Петровича Ершова //5-я международная конференция «Развитие вычислительной техники в России, странах бывшего СССР и СЭВ (SORUCOM 2020)». 2020. С. 87-101

3 П. Ершов Предварительные соображения о лексиконе программирования. 1983. Режим доступа: http://rkka21.ru/ershov-lexicon.htm

4 Городняя Л. В. Парадигма программирования. Учебное пособие // Лань Спб. 2019. ISBN 978-5-8114-3565-4. 232 с.

5 Floyd R. W. (1979). The paradigms of Programming Communications of the ACM 22 (8). 455p.

6 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

7 Городняя Л.В. Методика парадигмального анализа языков и систем программирования // XX Всероссийская научная конференция г. Новороссийск. М.:

ИПМ им. М.В.Келдыша. 2019. С. 262-277. Режим доступа: http://keldysh.ru/abrau/2019/theses/03.pdf DOI:10.20948/ abrau-2019-03

8 Лавров С.С. Методы задания семантики языков программирования // Программирование. 1978. № 6. С. 3-10.

9 Городняя Л.В. О представлении результатов анализа языков и систем программирования // XX Всероссийская научная конференция г. Новороссийск. М.: ИПМ им. М. В. Келдыша. 2018.

10 Городняя Л.В. Парадигмальная декомпозиция определения языка программирования //Научный сервис в сети Интернет: труды XVIII Всероссийской научной конференции г. Новороссийск. М.: ИПМ им. М.В. Келдыша. 2016. С. 115-127. Режим доступа: http://keldysh.ru/abrau/2016/21.pdf

11 Городняя Л.В., Кирпотина И.А. Методика декомпозиции сложных информационных систем // 30-я Международная конференция "Современные информационные технологии в образовании 2019" ИТО-Троицк-Москва. 2019 г. С.311-312

12 Лаврищева Е.М. Программная инженерия и технологии программирования сложных систем. Учебник для вузов. М. 2018 г. 432 с.

13 Maintenance. Диаграмма, представляющая хронологию появления и наследования многих ЯП. Режим доступа: https://www.levenez.com/lang/

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

15 Энциклопедия языков программирования. Режим доступа: http://progopedia.ru/

16 Peter Wegner. 1990. Concepts and paradigms of object-oriented programming. SIGPLAN OOPS Mess. 1. 1990. Pp. 7-87. . Режим доступа: https://pdfs.semanticscholar DOI:dx.doi.org/10.1145/

17 Городняя Л.В., Кирпотина И.А. О проблеме достоверности доступной в Интернете исторической фактографии // ТРУДЫ SORUCOM. 2017. С. 46-55. Режим доступа: https://www.sorucom.org/articles/materialy-mezhdunarodnoy-konferentsii-sorucom-2017/1648

18 Томилин А.Н. Информационные системы и научные телекоммуникации // Вестник РФФИ. 1998. № 4.

19 Городняя Л.В., Кирпотина И.А. Электронное издание как механизм самообучения // Тр. Междунар. научно-практической конф. «Новые информационные технологии в университетском образовании». Томск. 2000. С. 101–102

20 John von Neumann. First Draft of a Report on the EDVAC. University of Pennsylvania 1945.

21 Фуксман А.Л. Технические аспекты создания программных систем. М.: Статистика. 1979. 180 с.

22 Бабаев И.О., Новикова Ф.А., Петрушина Т.И. Язык ДЕКАРТ входной язык системы СПОРА. М.:Финансы и статистика. 1981. вып.1. С. 35-73

UDK 004.43 (042.4)

FROM DIFFICULT PROBLEMS TO PROGRAMMING PARADIGMS

Lidia V. Gorodnyaya
Candidate of Physical and Mathematical Sciences, Associate Professor, Senior Scientist,
e-mail: gorod@iis.nsk.su
Ershov Institute of Informatics Systems (IIS)
Siberian Branch of the Russian Academy of Sciences
630090 Novosibirsk, Academician Lavrentiev Avenue, 6.

Annotation. The article is devoted to the description and analysis of the relationship between hard-to-solve problems and the process of forming programming paradigms that can be useful in solving such problems.The material for analysis is provided by the history of computer technology, as well as the inconspicuous line of scientific activity of the leader of programming A.P. Ershov in the field of manifestation and development of programming paradigms.The results of the analysis made it possible to systematize programming languages, present an assessment of their similarities and differences, which allows constructing laconic definitions regarding paradigmatic models.This makes it possible to stratify the presentation of the peculiarities of the semantics of programming language 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, which can be useful for increasing software performance. Along the way, a ranking of problem statements is formed according to the degree of their knowledge and stading, which affects the assessment of labor intensity.

Keywords. systematization of programming languages, paradigm models, laconic definitions, A.P. Ershov, programming paradigm, strategic paradigms, programming problems, degree of study of problem statements, von Neumann architecture.

REFERENCES

1. Gorodnyaya L. V. Gumanitarnyye faktory programmirovaniya. Novosibirsk: SO RAN [Humanitarian factors of programming]. Novosibirsk: SB RAS = SB RAS. 2020. 163 p. ISBN 978-5-6044349-4-9 (in Russian)
2. Gorodnyaya L. V. Perspektivno strategicheskiye paradigmy programmirovaniya Akademika Andreya Petrovicha Yershova [Prospectively strategic programming paradigms of Academician Andrey Petrovich Ershov] // 5-ya mezhdunarodnaya konferentsiya «Razvitiye vychislitel'noy tekhniki v Rossii, stranakh byvshego SSSR i SEV (SORUCOM 2020)» = 5th international conference "Development of computer technology in Russia, the countries of the former USSR and CMEA (SORUCOM 2020)". 2020. Pp. 87-101.
3. P. Yershov Predvaritel'nyye soobrazheniya o leksikone programmirovaniya [Preliminary considerations about the vocabulary of programming]. 1983. Available at: http://rkka21.ru/ershov-lexicon.htm (in Russian).
4. Gorodnyaya L. V. Paradigma programmirovaniya. Uchebnoye posobiye [Programming paradigm. Textbook] // Lan' Spb = Lan SPb. 2019. ISBN 978-5-8114-3565-4. 232 p.
5. Floyd R. W. (1979). The paradigms of Programming .-- Communications of the ACM 22 (8). 455p.
6. Gorodnyaya L.V. Paradigmy programmirovaniya: analiz i sravneniye [Programming paradigms: 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
7. Gorodnyaya L.V. Metodika paradigmal'nogo analiza yazykov i sistem programmirovaniya [Methods of paradigm analysis of languages and programming systems] // XX Vserossiyskaya nauchnaya konferentsiya g. Novorossiysk. M.: IPM im. M.V.Keldysha = XX All-Russian scientific conference Novorossiysk. M .: IPM im. M.V. Keldysh. 2019. Pp . 262-277. Available at: http://keldysh.ru/abrau/2019/theses/03.pdf DOI: 10.20948 / abrau-2019-03.

8. Lavrov S.S. Metody zadaniya semantiki yazykov programmirovaniya [Methods for setting the semantics of programming languages] // Programmirovaniye = Programming. 1978. № 6. Pp. 3-10.
9. Gorodnyaya L.V. O predstavlenii rezul'tatov analiza yazykov i sistem programmirovaniya [On the presentation of the results of the analysis of languages and programming systems] // XX Vserossiyskaya nauchnaya konferentsiya g. Novorossiysk M.: IPM im. M. V. Keldysha = XX All-Russian Scientific Conference Novorossiysk M .: IPM im. M.V. Keldysh. 2018.
10. Gorodnyaya L.V. Paradigmal'naya dekompozitsiya opredeleniya yazyka programmirovaniya [Paradigmatic decomposition of programming language definition] //Nauchnyy servis v seti Internet: trudy XVIII Vserossiyskoy nauchnoy konferentsii g. Novorossiysk. M.: IPM im. M.V. Keldysha = Scientific service on the Internet: proceedings of the XVIII All-Russian scientific conference in Novorossiysk. M .: IPM im. M.V. Keldysh. 2016. Pp. 115-127. Available at: http://keldysh.ru/abrau/2016/21.pdf.
11. Gorodnyaya L.V., Kirpotina I.A. Metodika dekompozitsii slozhnykh informatsionnykh sistem [Methodology for decomposition of complex information systems] // 30-ya Mezhdunarodnaya konferentsiya "Sovremennyye informatsionnyye tekhnologii v obrazovanii 2019" ITO-Troitsk-Moskva = 30th International Conference "Modern Information Technologies in Education 2019" ITO-Troitsk-Moscow. 2019. Pp. 311 – 312.
12. Lavrishcheva Ye.M. Programmnaya inzheneriya i tekhnologii programmirovaniya slozhnykh sistem. Uchebnik dlya vuzov [Software engineering and technologies for programming complex systems. Textbook for universities] // M. 2018. 432 p.
13. Maintenance. Diagramma, predstavlyayushchaya khronologiyu poyavleniya i nasledovaniya mnogikh YAP. Available at: https://www.levenez.com/lang/
14. Peter Van Roy. Diagram showing the results of comparing more than 30 programming paradigms. Available at: https://www.info.ucl.ac.be/~pvr/ paradigmsDIAGRAMeng108.pdf
15. Entsiklopediya yazykov programmirovaniya [Encyclopedia of programming languages]. Available at: http://progopedia.ru/
16. Peter Wegner. 1990. Concepts and paradigms of object-oriented programming. SIGPLAN OOPS Mess. 1. 1990. Pp. 7-87. Available at: https://pdfs.semanticscholar DOI:dx.doi.org/ 10.1145/
17. Gorodnyaya L.V., Kirpotina I.A. O probleme dostovernosti dostupnoy v Internete istoricheskoy faktografii [On the problem of authenticity of historical factography available on the Internet] // TRUDY SORUCOM = SORUCOM PROJECTS. 2017. Pp. 46-55. Available at: https://www.sorucom.org/articles/materialy-mezhdunarodnoy-konferentsii-sorucom-2017/1648
18. Tomilin A.N. Informatsionnyye sistemy i nauchnyye telekommunikatsii // Vestnik RFFI = RFBR Bulletin. 1998. № 4.
19. Gorodnyaya L.V., Kirpotina I.A. Elektronnoye izdaniye kak mekhanizm samoobucheniya [Electronic publication as a self-learning mechanism] // «Новые информационные технологии в университетском образовании» = Tr. Int. scientific and practical conf. "New information technologies in university education". Tomsk. 2000. Pp. 101-102.
20. John von Neumann. First Draft of a Report on the EDVAC. University of Pennsylvania 1945.
21. Fuksman A.L. Tekhnicheskiye aspekty sozdaniya programmnykh sistem [Technical aspects of creating software systems] / M.: Statistika = M .: Statistics. 1979.180 p.
22. Babayev I.O., Novikova F.A., Petrushina T.I. YAzyk DEKART vkhodnoy yazyk sistemy SPORA [The DECART language is the input language of the SPORA system. M.: Finance and statistics]. 1981. issue 1. Pp. 35-73.