Подход к оценке трудоёмкости программирования

Л.В. Городняя

Институт систем информатики СО РАН

Аннотация. Доклад посвящен вопросам, возникающим в связи с проблемой измерения характеристик языков программирования, влияющих на трудоёмкость разработки программного обеспечения. Парадигмальные модели языков и систем программирования могут быть полезны не только при систематизации языков программирования, оценке их сходства и различия, но и при формировании оценок внешней, учебной и внутренней, реализационной сложности, учёт которой необходим при прогнозировании хода процессов применения, планировании изучения и организации разработки программ.

Ключевые слова: измеримые характеристики, парадигмальные модели, преподавание системного программирования, лаконичные определения

An approach to the measure of the labour required for programming

LV. Gorodnyaya

A.P. Ershov Institute of Informatics Systems (IIS), SB RAS

Abstract. The report is devoted to issues arising in connection with the problem of measuring the characteristics of programming languages that affect the labour required for software development. Paradigmatic models of programming languages and systems can be useful not only in the systematization of programming languages, in assessing their similarities and differences, but also in the formation of assessments of external, educational, and internal complexity, taking which into account is necessary to predict the course of application, study, and program development.

Keywords: paradigm models, autonomously developed components, teaching system programming, concise definitions

Программирование давно уже стало массовой профессией, требующей объективных метрик для оценки качества программируемых

решений. Препятствует формированию таких метрик слишком высокий темп развития ИТ-индустрии, при котором доминируют быстрые интуитивные или волевые решения, дающие экономические выигрыши. Другое препятствие связано с двойственностью оценки сложности программ, точнее с разночтениями в оценке сложности применения и сложности разработки программ — **внешняя и внутренняя** сложность программ. Границу между ними трудно осознавать.

В докладе представлена попытка предложить методику измерения характеристик языков программирования (ЯП) на основе оценки понятийной сложности в форме, позволяющей прогнозировать трудоёмкость программирования и благодаря этому ориентироваться в современном пространстве средств и методов, определяющих дальнейшее развитие ИТ. В первом разделе отмечается уязвимость интегральности непосредственных измерений трудоёмкости программирования в таких метриках как «человеко-месяц» потому, что не выявлено положительной зависимости производительности программ от трудоёмкости программирования [1,2]. Во втором разделе рассматриваются легко удостоверяемые измеримые параметры трудоёмкости программирования и производительности программ. Большинство из них почти не отражают зависимости результата программирования от принятых программистом решений и выбора конструкций ЯП [3]. В третьем разделе анализируется связь трудоёмкости программирования и степени изученности решаемых задач. Недоучёт такой связи обычно приводит к систематическим ошибкам прогноза предстоящих трудозатрат [4].

В четвёртом разделе отмечена роль квалификации программиста, показывающая актуальность разработки методик измерения важных для практики характеристик программ, а также создания системы непрерывной переподготовки кадров, квалификация которых должна успевать за темпом прогресса ИТ-технологий. Пятый раздел посвящен актуальности вопросов, связанных с удостоверением профессиональной квалификации программистов. Следующий за ним шестой раздел содержит перечень выводов из обсуждения относительно места системного программирования в сфере ИТ-индустрии, подходов к образованию программистов и особенно системных программистов [6]. Далее, в седьмом разделе предлагается рассмотреть понятийную сложность средств программирования. Её можно оценвать в рамках парадигмально семантической декомпозиции, результаты которой представляются понятийными таблицами, способными работать примерно как визитная карточна языка или системы программирования. Позициям такой таблицы можно сопоставить элементы ознакомительного ряда отлаженных примеров, иллюстрирующих смысл понятий ЯП [4,5].

В заключении выражена гипотеза, что формулу трудоёмкости программирования можно формализовать и сделать понятной, привлекая

для характеристики элементов понятийной таблицы некоторые образы из массово используемых информационных систем.

1. Интегральность измерений при оценке трудоёмкости

Ещё Фр. Брукс в начале 1970-х годов отмечал уязвимость оценки трудоёмкости программирования в таких метриках как «человеко-месяц» или «человеко-день» [1,2]. Мало того, что производительность труда отдельных людей много от чего зависит, так и за месяц непредсказуемо многое может произойти. В этом плане интересен эффект, проявившийся при массовом производстве сайтов в 1990-е годы, которое тогда можно было пронаблюдать непосредственно. В этой сфере удавалось работы структурировать так, что каждый исполнитель получал задание утром на один день, лишь иногда — на неделю. При такой схеме производство шло достаточно стабильно. Жаль только, что ниша эта быстро исчерпалась.

Другой аспект — зависимость трудоёмкости и качества программирования от квалификационного уровня программистов. По свидетельству Дж.Вейнберга, автора уникальной монографии «Психология программирования» [8] фирма ИБМ в начале 1970-х провела исследования разброса трудоёмкости программирования в зависимости от опыта, возраста и способностей. Оказалось, что на простых задачах разброс составляет 1 к 28, причём почти независимо от опыта и возраста. На сложных задачах заметна положительная зависимость от способностей, опыта и возраста с несколько меньшим разбросом, примерно 1 к 10. Причём скорость выполнения заданий редко способствует качеству решений, чаще наоборот, скороспелые решения при программировании приводят к много большим трудозатратам при отладке и эксплуатации программ. Не меньший разброс был замечен и в оценки качества программируемых решений, не выявлено положительной зависимости производительности программ от трудоёмкости программирования.

Экономисты объясняют отсутствие методики измерения и прогнозирования трудоёмкости программирования тем, что программистские проекты слишком разнообразны, каждый проект уникален, в них трудно видеть типовые схемы. Можно вспомнить страсти 1980-90-ых годов в дискуссиях о приоритетах между вопросами «**Что?**» и «**Как?**». На их фоне не произошло понимания, что самым главным является вопрос «**Кто?**». Кто способен замечать новые задачи и открывать их практичные решения с помощью программ? Не исключено, что ответ именно на этот вопрос даёт подход к типизации программистских проектов.

2. Параметры трудоёмкости программирования

Следует отметить, что жаргон современного практического программирования использует понятие «язык программирования» как

«входной язык – расширенное подмножество языка программирования типовой системы программирования (СП), 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 на практике различий между языками и системами программирования (ЯиСП). Кроме того, прямые измерения трудоёмкости программирования и производительности программ почти не отражают зависимости результата от принятых программистом решений и выбора конструкций ЯП. Хотя программируемые решения представляются в терминах ЯП, его влияние растворяется в весьма сложном комплексе, наследующем производительность СП и оборудования. Таким образом, существует проблема создания методики, позволяющей выявлять такие зависимости совмещением прямых измерений с результатами экспертных оценок особенностей ЯиСП, возможно отличающихся от оценок ЯП.

Есть основания при прогнозировании трудоёмкости программирования учитывать **степень изученности** решаемых задач, **уровень квалификации** и способностей разработчиков программы решения задачи и **понятийную сложность** реализуемых и используемых программируемых и программных средств. Вопрос оценки сложности систем программирования был подробно исследован Т.С.Васючковой, убедительно показавшей, что сравнение СП требует более чем 200 параметров [3]. Представление результатов оценки понятийной сложности, позволяющее структурировать пространство таких параметров, рассмотрено в статье [4].

3. Степень изученности решаемых задач

По степени изученности существенно различаются следующие пространства постановок задач, влияющие на выбор методов решения задач и трудоёмкость их программирования:

- новые;
- исследовательские;
- практичные;
- точные.

Для **новых** постановок задач характерно отсутствие доступного прецедента решения задачи, новизна используемых средств или неопытность исполнителей. **Исследовательск**ие постановки задач обычно усложнены требованиями оригинальности и универсальности, допускающими демонстрацию в эксперименте. **Практичные** постановки задач нацелены на актуальность и удобство многократного применения. **Точные** постановки задач включают в себя испытание предельных возможностей используемых средств, связанных с мерой организованности

созданной программы и рангом работоспособности реализованных решений. Разброс трудоёмкости в зависимости от степени новизны или изученности постановки задачи обычно составляет примерно 1 к 8. Недоучёт такого разброса обычно приводит к систематическим ошибкам прогноза предстоящих трудозатрат.

Макетный образец решения новой задачи работоспособен при предъявлении автором небольшого набора подходящих данных. Для экспериментального полигона отладки решений исследовательских задач требуется приспособленность к обработке почти любых данных, которые могут быть даны в качестве входных. Практичная версия может быть ограничена обработкой данных, реально встречающихся в сфере её приложения. Для точной реализации нужны специально подобранные данные, показывающие её превосходство над менее точно и эффективно выполненными решениями задачи, зато она обычно может использовать готовые алгоритмы и прототипы.

4. Уровни квалификации

При анализе квалификации программистов можно рассмотреть и сравнить содержательную сложность трудовых функций специалистов одного уровня, представленную в системе профессиональных стандартов. Для примера рассмотрим 6-й уровень стандарта на профессию «Программист» и стандарт на профессию «Системный программист». Разница сосредоточена на требованиях, нацеливающих системного программиста на понимание смысла решаемых задач, развитие области приложения решений, определение границы применимости полученных результатов и достижение системности при обобщении созданных инструментальных средств, важнейших для решения задач современных ИТ.

Результаты программирования, интегрированные как ИТ-индустрия, справедливо объясняются системным подходом к решению массово востребованных задач. Характерной чертой системного подхода как ведущего метода программирования является переход к классу задач при содержательном анализе отдельных конкретных постановок задач. Границы класса устанавливаются выбором процесса решения задач. Результаты содержательного анализа в конце концов овеществляются в форме достаточно удобной для их массового применения и заимствования: программная документация, языки программирования, системы команд компьютера – архитектура, операционные системы, системы программирования, программные инструменты, библиотеки программ, алгоритмы, руководства по применению. Это позволяет переходить к массовому применению готовых модулей или при сборке решений аналогичных задач обходиться без глубокого анализа и претензий на понимание каждой задачи и каждого модуля.

Все это не представляет собой ничего особенного, если не обратить внимание на то, что применение компьютеров и телекоммуникаций при реализации программ, порождающих механизмов и решений конкретных задач допускает соблазнительную возможность вместо непосредственного решения задач, стремительно переходить к практике применения готовых рецептов независимо от трудозатрат на достижение хорошего понимания задачи. Симптоматична рекомендация лёгкость применения нейронных сетей не обременять попытками понимания логики решений. Результат увлечения этой возможностью общеизвестен – избыточная сложность программного обеспечения, обуславливающая неудовлетворительную надежность его применения и рост накладных расходов на его изучение, сопровождение и перенос на новые архитектуры. Попытки преодоления этого неприятного эффекта аналитическим путем привели к постановке не менее сложных проблем. Это можно рассматривать как обоснование необходимости более фундаментального подхода к программированию, особенно к системному программированию. Поэтому подготовка специалистов в такой сфере требует налаженного взаимодействия университетов с наукой и ИТ-индустрией, потребляющей результаты экспериментально исследовательской активности, включая разработку необходимых методик измерения важных для практики характеристик программ и создание системы непрерывной переподготовки кадров.

5. **Актуальность проблемы**

Обсуждение вопросов, связанных с удостоверением профессиональной квалификации программистов, состоялось в рамках программы 12-й Ершовской международной конференции «Перспективы систем информатики» (ПСИ-19), проведённой в Академгородке Новосибирска, 3 июля 2019 года в форме объединённого круглого стола объединивший работу всех трёх рабочих семинаров Конференции: «Наукоёмкое программное обеспечение», «Информатика образования» и «Семантики программ, спецификация и верификация», участники которых — представители образования, науки и ИТ-индустрии. [6] Среди многих были рассмотрены следующие темы:

– вопросы перспектив системного и прикладного программирования;

– актуальность создания средств программирования, исключающих на практике обилие непринципиальных ошибок в программах, поддерживающих понимание всех подходов к созданию программ и программных систем;

– сокращение разрыва между исторически сложившимися программами подготовки программистов и международными требованиями к таким программам, обогащение таких требований наследованием успешного отечественного опыта;

– прецедент успешного образования программистов под Ершовским лозунгом «Программирование — вторая грамотность» формировался шире, чем чисто университетское обучение [1], дополнением было и самообразование со взаимообучением, и дистанционное общение, и факультативные формы типа Летних школ, проводимых ИСИ СО РАН;

– иллюзия насыщения потребности в ИТ-специалистах, особенно в системных программистах, нередко насаждаемая менеджерами среднего звена, которых раздражает склонность программистов уточнять постановки задач;

– тенденция перепрофилирования программистов в грамотных пользователей или менеджеров;

– кадровый дефицит в ИТ-индустрии, потребность в специалистах с уникальными навыками системного программирования, применимыми у конкретного работодателя.

6. Выводы из обсуждения

В результате обсуждения сделаны конструктивные выводы относительно места системного программирования в сфере ИТ-индустрии, подходов к образованию программистов и особенно системных программистов, вопросов формирования и исследования математических моделей как фундаментального базиса для решения особо важных проблем эффективности, надёжности и безопасности программного обеспечения и перспектив организации системы поддержки квалификационного и карьерного роста программистов, включая системных программистов:

– Востребованность системного программирования как ключевого механизма решения важных задач ждёт своего описания, определяющего круг проблем, решаемых экспериментальной, системной и теоретической информатикой, дающих основания для перехода к инженерным решениям, отвечающим вызовам компьютерных технологий. В числе таких задач следует отметить задачи системного программирования по созданию конструктивных средств программирования, достаточных для общего владения различными подходами к созданию надёжных и безопасных программ в разных областях приложения.

– При создании, формировании и исследовании математических моделей как фундаментального базиса для решения особо важных проблем

[1] Следует учесть существенную для программирования разницу между понятиями «обучение» и «научение». Научение проявляется в приобретённых навыках, которые можно увидеть в изменении поведения. Обучение нацелено на получение знаний, которые могут косвенно проявляться в когнитивных процессах, а могут и никак не проявляться при экзаменационной проверке. И наоборот, видимое проявление знаний может никак не проявляться в практических навыках. Программирование, как любая практическая деятельность, больше зависит от научения, чем от обучения.

эффективности, надёжности и безопасности программного обеспечения следует обратить особое внимание на развитие моделей, связанных со временем и ресурсами, слабо представленных в курсах по классической математике.

– Квалификация системного программиста включает в себя глубокое владение фундаментальной математикой, ознакомление с современной физикой, способность самостоятельно осваивать незнакомое новое и передавать полученное знание коллегам, склонность изобретать свои решения новых задач и навыки повышать качество готовых своих и чужих решений, что требует умения отличать правильное от ложного и разумное от нерационального.

– Математические основы позволяют сформировать **метрическое пространство** для методики оценки сложности программирования, дающей базу для прогнозирования трудоёмкости применения и разработки программ.

– Экстенсивное развитие ИТ заметно опережает возможности человека оперативно осваивать новые возможности аппаратуры и системных средств ИТ, выходящие за пределы пользовательского уровня, поддерживаемого поставщиками инструментария и программных продуктов. Поэтому необходима система повышения квалификации и карьерного роста, нацеленная на регулярное ознакомление практиков, квалификация которых обрела жизненно важное для общества значение, с новыми возможностями ИТ. Такая система может удостоверять и поощрять квалификационный рост специалистов на протяжении всей профессиональной деятельности в области практического программирования, научно-исследовательской активности и системного программирования.

– В современных условиях расширения зависимости всех сфер жизнедеятельности от автоматизации на базе ИТ-технологий следует обратить внимание на роль квалификации ИТ-специалистов в области **системного программирования**, возможности карьерного роста в этом направлении и поддержки поствузовского повышения квалификации специалистов в этой **жизненно важной системе общества, сравнимой с медициной и образованием**. Следует особо отметить мировую тенденцию заимствования прикладным программированием средств и методов системного программирования, выявившуюся в форме массового создания проблемно ориентированных компьютерных языков (DSL).

– Нередко системные программисты потребность в карьерном росте реализуют переходом в технические писатели, руководители проектов или менеджеры. Это чревато не только накладными расходами на овладение иной профессией, но и утратой навыков практического программирования, обусловленной высоким темпом смены элементной базы и пользовательских интерфейсов.

– В пространстве ИТ-индустрии профессия "Системный программист" не нацелена непосредственно на производство товара массового спроса, что затрудняет оценку вклада системного программирования в экономику ИТ-технологий. **Миссия системного программирования** – создание инструментария для решения новых задач, иногда неожиданных, а также для поддержки методов повышения качества информационных систем, включая поиск решений по обеспечению надежности и безопасности информационных технологий.

Общее решение образовательных проблем программирования можно резюмировать следующим образом: В стремительно обновляющемся мире ИТ-технологий главная цель обучения будущих программистов — вывод их на уровень перехода к самообучению и изобретению решений новых, ранее не известных, задач. Это освободит учебный процесс от гонки за производственными технологиями, устаревающими раньше завершения учёбы, и даст путь к оперативному освоению новинок без избыточных трудозатрат на забывание неактуального или устаревшего. Известно, что первичное научение в 3-5 раз легче и надёжнее, чем перестройка старых навыков.

7. Понятийная сложность средств программирования

Представление результатов анализа ЯП можно рассматривать как основу для измерения понятийной сложности [4]. Для этого достаточно изучение описаний ЯП сопровождать парадигмально семантической декомпозицией, результаты которой представляются понятийными таблицами. Из таких таблиц можно вывести матрицу численных характеристик понятийной сложности ЯП. Позициям такой таблицы можно сопоставить элементы ряда учебных примеров, иллюстрирующих смысл понятий ЯП. Для мультипарадигмальных ЯП можно представить несколько понятийных таблиц по числу поддержанных парадигм. Пример использования понятийных таблиц при сравнении ЯП приведён в Приложении 1, Таблица 1. Наполнение понятийных таблиц разными экспертами могут отличаться. Студенты заполняют понятийную таблицу для знакомого им языка за 4-6 часов. Такие **понятийные матрицы достаточны** для представления результатов анализа ЯП и грубой оценки внешней сложности их применения при прикладном программировании, например, по числу различимых конструкций в каждой клетке понятийной матрицы. Более тонкая оценка внутренней сложности реализации ЯиСП потребует комплекса таких матриц, показывающих сложность внутренних построений, таких как тексты, внутренние форматы, структуры, коды.

Для более точной оценки ЯиСП и анализа измеримых характеристик можно использовать комплекты сопоставимых фрагментов программ на оцениваемых ЯП в форме, приспособленной для прямого эксперимента с СП, поддерживающих изучаемые ЯП. Ряд иллюстративных примеров

может отражать методику обучения. Примеры накапливаются как упорядоченный по сложности их усвоения обучающий ряд. Каждый пример иллюстрирует особенности использования отдельного понятия. Для иллюстрации некоторых понятий может требоваться несколько примеров. Последовательность элементов ряда соответствует порядку изучения понятий, подкреплённому компьютерной практикой. Каждый 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 внешнюю 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. Пример использования иллюстративных примеров для более детального сравнения ЯП приведён в Приложении 1, Таблица 2.

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 обычно определены списки поддержанных в нём парадигм, предшественников, сфера влияния. Такие характеристики можно выводить и уточнять из списков общих семантических систем, оценивая существенную разницу в традиционной прагматике функционально эквивалентных систем.

8. Заключение. Формула трудоёмкости

Имеется не проверенная на больших проектах гипотеза, что общую формулу оценки трудоёмкости можно выразить как сумму произведений оценок **степени изученности** решаемой задачи (от 8 до 1), **квалификации программиста** (от 1 до 28) и **понятийной сложности** основных работ, связанных с разрабатываемой программой. В этой тройке особенно непонятен вывод оценки **квалификации** программиста, зависящей не только от образования и опыта, но очень явно от способностей и интуитивных механизмов, проявление которых может сильно противоречить традициям общеобразовательной системы, нацеленной на контроль статического знания вместо поощрения динамики и перспектив развития, а также методикам управления проектами, рекомендующих работать без «звёздных» программистов. Тем не менее среди практиков обычно формируется вполне достоверная интуитивная оценка профессионального потенциала знакомых программистов.

Понятийная **сложность** представляет собой функцию от внешних и внутренних оценок, допускающую нормирование на интуитивно понятную оценку простых работ, выполняемых и непрофессионалами, и

специалистами. Важно, чтобы **нормирующий вид работы** был образно и интуитивно понятен и специалисту и пользователю, подобно метру или грамму. Примерно в таком стиле представляют научные данные для популяризации результатов физики, биологии и т.д. Так например, для оценки трудоёмкости интерфейса программы (число кнопок, пунктов меню, позиций настройки) роль нормирующего коэффициента может выполнять интерфейс торгового автомата, подобного автомату А.Хоара (1-4 клавиши). С таким автоматом справляется практически любой человек. Нормировочным коэффициентом трудоёмкости ряда учебных примеров может быть символьная подстановка, привычная большинству по работе с бланками (2-8 команд). Интерфейс и обучающий ряд можно рассматривать как внешнюю, поверхностную характеристику программы. Устройство знакомого со школьных времён калькулятора может выполнять роль нормирующего коэффициента для вывода внутренней, глубинной сложности понятийной структуры программы.

Типичная семантическая система обычно сравнима по сложности с калькулятором или автоматом или с обработкой бланков. Например, внешняя сложность применения программы «Блокнот — Windows 10» – 34 пункта меню, что можно оценить как эквивалент 10-ти торговых автоматов. Внешняя учебная сложность введения в язык PureLisp — 240 упражнений, можно оценить как 30 бланков разного формата (символьные вычисления). Внутренняя сложность реализации языка Little, подобного упрощённой версии языка Fortran — более 430 функций, можно оценить как взаимодействие 80-ти калькуляторов.

Таким образом, выстраивается подход, позволяющий учитывать особенности решаемых задач, влияющие на выбор методов их решения, в зависимости от приоритетов в выборе языковых средств и реализационных конструкций [5]. Намечена линия, позволяющая сравнивать языки, выделяя сопоставимые примеры программ, и анализировать результаты прямых измерений, производительности программ, выделяя особенности базовых средств и реализационных решений в системах программирования [7]. Примеры использования понятийных таблиц и иллюстрирующих фрагментов программ при сравнении языков программирования приведены в Приложении 1.

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

Литература

1. Брукс Ф. П. мл. Как проектируются и создаются программные комплексы. (Серия: «Библиотечка программиста») — М.: Наука, 1979. — 152 с илл.

2. Брукс Ф. П. мл. Мифический человеко-месяц. – М.: Символ-Плюс, 2000. – 304 с.
3. Васючкова Т. С. Методы измерения качества программного обеспечения. – Препринт N 339, Новосибирск, ВЦ СО АН СССР, 1981, 30 с.
4. 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. С. 262-277. URL: https://doi.org/10.20948/abrau-2019-03
5. Городняя Л. В. О классификации парадигм и языков программирования. // 12-ая Международная Ершовская конференция по информатике (PSI'2019), семинар "Наукоемкое программное обеспечение (НПО)" Новосибирск, 2-5 июля 2019 года, с. 83-89
6. Лаврентьев М. М., Городняя Л. В., Держо М. А., Мигинский Д. С., Вопрос карьерных перспектив в области ИТ // 12-ая Международная Ершовская конференция по информатике (PSI'2019), семинар "Наукоемкое программное обеспечение (НПО)" Новосибирск, 2-5 июля 2019 года, с. 111-120
7. Городняя Л. В., Демидов С. В., Кириченко М. Д., Ткаченко Д. Д. Проект информационного стенда ПРИЗМА для представления измеримых характеристик языков и систем программирования // Иркутск, «Информационные и математические технологии в науке и управлении», с. 105-119 . eLIBRARY ID: 42876216, DOI: 10.38028/ESI.2020.17.1.008
8. Weinberg G. M. The Psychology of Computer Programming. – New York: Van Norstand Reinhold Comp., 1971.

References

1. Bruks F. P. ml. Kak proyektiruyutsya i sozdayutsya programmnyye kompleksy. (Seriya: «Bibliotechka programmista») — M.: Nauka, 1979. — 152 s ill.
2. Bruks F. P. ml. Mificheskiy cheloveko-mesyats. – M.: Simvol-Plyus, 2000. – 304 s.
3. Vasyuchkova T. S. Metody izmereniya kachestva programmnogo obespecheniya. – Preprint N 339, Novosibirsk, VTS SO AN SSSR, 1981, 30 s.
4. Gorodnyaya, L. V. O predstavlenii rezul'tatov analiza yazykov i sistem programmirovaniya. Nauchnyy servis v seti Internet: trudy XX Vserossiyskoy nauchnoy konferentsii (17-22 sentyabrya 2018 g., g. Novorossiysk). — M.: IPM im. M. V. Keldysha, 2018. S. 262-277. URL: https://doi.org/10.20948/abrau-2019-03

5. Gorodnyaya L. V. O klassifikatsii paradigm i yazykov programmirovaniya. // 12-ay Mezhdunarodnay Yershovskay konferentsiya po informatike (PSI’2019), seminar "Naukoyemkoye programmnoye obespecheniye (NPO)" Novosibirsk, 2-5 iyulya 2019 goda, s. 83-89
6. Lavrent'yev M. M., Gorodnyaya L. V., Derzho M. A., Miginskiy D. S., Vopros kar'yernykh perspektiv v oblasti IT // 12-ay Mezhdunarodnay Yershovskay konferentsiya po informatike (PSI’2019), seminar "Naukoyemkoye programmnoye obespecheniye (NPO)" Novosibirsk, 2-5 iyulya 2019 goda, s. 111-120
7. Gorodnyaya L. V., Demidov S. V., Kirichenko M. D., Tkachenko D. D. Proyekt informatsionnogo stenda PRIZMA dlya predstavleniya izmerimykh kharakteristik yazykov i sistem programmirovaniya // Irkutsk, «Informatsionnyye i matematicheskiye tekhnologii v nauke i upravlenii», s. 105-119 . eLIBRARY ID: 42876216, DOI: 10.38028/ESI.2020.17.1.008
8. Weinberg G.M. The Psychology of Computer Programming. – New York: Van Norstand Reinhold Comp., 1971.

Приложение 1

Таблица 1.
Результаты сравнения двух ЯП на уровне базовой семантики

Индекс позиции [2]	*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*
				*Диаграммы доступа к примерам* ***(визитная карточка)***
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 в перечень *- потеряно* — возможность определять смысл по ходу дела *+ добавлено* — чёткость смысла
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 соответствующими им значениями. Адреса и таблица могут быть	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

[2] Индекс имеет формат: XY, где X — обозначает строку понятийной матрицы, а Y — столбец, оба имеют значения из E, M, C, S.

	значениями или не рассматриваться как значения и использоваться неявно как сущности другой природы. Появляется именование значений и формул, что расширяет класс допустимых формул, обеспечивает многократное использование хранимых результатов и приводит к понятию «данное».			данных, а Паскаль требует не только понятия “распределение памяти под идентификаторы”, но они ещё разделены на метки и переменные, причём организацтя 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 *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 в Паскае инфиксные - *потеряно* — свобода от учёта приоритетов операций + *добавлено* — наследование привычек к принятым в математических 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 имя *≠ различие* - Лисп сводит структуры данных к парам, а Паскаль к блоку соседних регистров - *потеряно* — автоматизация повторного использования памяти GC *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.

Таблица 2

Сравнение фрагментов уровня ядра языка

Индекс [3]	*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 2+3	С точностью до алфавита
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

[3] Индекс имеет формат: XYz, где z – номер пункта в понятийной матрице, X — обозначает строку понятийной матрицы, а Y — столбец, X и Y - из E, M, C, S.

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;
		h:= h/2;
			(EVAL (CONS T NIL))
			(QUOTE A)
			(QUOTE (EVAL A))
			(EVAL (QUOTE A))
				С точностью до порядка вычислений

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))** 2. 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 [1 . 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.			