# Глава 11. **Эволюция развития языков и систем программирования**

Язык формирует наш способ мышления и
определяет, о чем мы можем говорить.
Б.Л. Ворф

## 11.1. Состояние проблемы

В настоящее время насчитывается несколько сотен различных языков программирования. Как известно [1], для повышения эффективности (производительности, отказоустойчивости и т.д.) вычислительных средств необходимо совершенствовать элементно-технологический, информационный, организационный и алгоритмический базисы. Эффективность машинных алгоритмов во многом зависит от используемых языков и систем программирования. При этом повышение эффективности языков и систем программирования тесно связано с совершенствованием следующих средств: анализ и обработка информации; информационное обеспечение, организация обработки и управления данными, процедурами, моделями и т.д., декомпозиция больших программ, семантических, синтаксических и морфологических возможностей языка; адаптация к внутренним и внешним условиям использования языка; технология программирования и т.п. Одним из средств адаптации языков программирования является повышение уровня абстрагируемости данных, который повышает их мощность, гибкость, прозрачность и транспортабельность. Другие средства адаптации направлены на открытость уровня средств обработки данных, учет особенностей технических средств при компилировании программы (учет возможностей параллельной работы, расширение существующих в процессоре конвейеров и средств внутреннего планирования выполнения инструкций и т.д.). Особое место в адаптации к внешним условиям применения языков и систем программирования играет повышение их интеллектуального уровня. Изучение эволюции языков и систем программирования невозможно одновременно во всех направлениях. Поэтому изучение этих процессов будет представлено в разных проекциях. Наиболее важные связаны с рассмотрением различных точек зрения: предметных областей, данных и структур, декомпозиции элементов, статики и динамики описания моделей, процессов, технологии программирования, уровня интеллектуализации и т.д.

## 11.2. Классификация и сопоставление языков программирования

Существуют различные классификации языков программирования. Наиболее общей является классификация по степени зависимости языка от компь-

ютера. По этой классификации языки программирования делят на два больших класса: машинно-зависимые и машинно-независимые. В свою очередь, машинно-зависимые языки делят на машинные (инструкции компьютера) и машинно-ориентированные (языки символьного кодирования, макроязыки и т.д.). В данной работе эти классы языков рассматриваться не будут. В свою очередь, машинно-независимые языки делятся на процедурные, иногда называемые операторными или императивными, и декларативные. При этом необходимо отметить, что, по своей сущности, большинство ныне существующих языков являются комбинированными либо смешанными и классифицируются по основным средствам, ими используемым. В этом смысле они подчинены одному из общих законов эволюции – закону смешанного экстремума [2]. Так, ряд машинно-независимых языков высокого уровня делают таким образом, чтобы выражения ветвления в той или иной степени соответствовали номенклатуре машинных инструкций и способам адресации процессора (PL/65, Mistral, Smal/80, PL/360) и др. Далее будет показано, что подобная "смешанность" присуща также императивным и декларативным языкам. Большинство из ныне используемых языков программирования относится к процедурным языкам (Бейсик, Си, Фортран, Алгол, Кобол, Паскаль, Ада и многие др.).

Наиболее существенными классами декларативных языков являются функциональные, называемые также аппликативными, и логические. К категории функциональных языков относятся: Лисп, FP, Apl, Nial, Krc, Logo и другие, а логического программирования -Пролог и другие.

В работе [3] справедливо отмечается: "На практике языки программирования не являются чисто процедурными, функциональными или логическими, а содержат в себе черты языков различных типов. Это утверждение полностью соответствует принципу смешанного экстремума [2], являющемуся одним из фундаментальных принципов природы. С точки зрения практики, это приводит к тому, что на процедурном языке можно написать функциональную программу либо ее часть и наоборот. Все это связано с тем, что большинство языков высокого уровня учитывают разные стили и методы программирования.

Отметим, что в функциональном программировании высший приоритет отдается понятию функции и предполагает жесткое структурирование данных (последовательность последовательностей), что создает сложности для представления привычных для программистов данных в виде массивов, записей и тому подобных структур данных.

В свою очередь, логическое программирование акцентирует внимание на средствах логики. Но эти средства ограничены возможностями чисто декларативных логик.

Была сделана попытка объединения логического программирования с функциональным. Примером такого подхода является создание языка Логлисп, который реализует парадигму логического программирования в рамках языка

Лиспа, что еще раз иллюстрирует действие вышеупомянутого принципа смешанного экстремума.

По большому счету процедурные языки в большей своей части ориентированы на решение вычислительных и близких к ним задач, а декларативные – на невычислительные задачи и задачи искусственного интеллекта.

Мы рассмотрели две крайности, но в действительности спектр языков программирования имеет и множество полутонов, на которых остановимся в дальнейшем.

### 11.3. Эволюционное развитие языков и систем программирования

Языки и системы программирования подчинены общим законам эволюции. О всеобщности этих законов для живой, неживой и искусственной природы уже писалось в работах автора [2, 4, 5]. Но в каждой области исследования эволюция имеет как общие черты, так и свою специфику. Общим является обеспечение адаптации к внешним и внутренним условиям применения, испытание исследуемых объектов на выживание на основе основного метода природы и инженеров – проб и ошибок.

Специфическим в данном исследовании является то, что в нем доминируют два аспекта: развитие технологии вычислительных средств и программирования. В обоих этих аспектах главенствует творчество человека. Но если первый аспект связан с материальным производством, то второй, в основном, – с интеллектуальной деятельностью человека. Это не означает, что в достижениях материального производства не участвует интеллектуальная деятельность, а говорит только о том, что существенные ограничения на прогресс материального производства имеют материальные факторы.

Эволюция происходит во времени и пространстве и в большинстве случаев ростки будущего надо искать в прошлом. Но для искусственных сил, особенно интеллектуальных, нередко случается, что некоторые идеи и даже конкретные объекты творчества могут намного опережать свое время. Однако они могут только приблизить будущее, но не нарушить общий ход эволюционного процесса. Психологическое и философское обоснование этих явлений оставим специалистам других областей. Нас же интересуют более конкретные вопросы. Какие параметры языков и систем программирования наиболее существенны для эволюционного развития? На основе каких законов можно прогнозировать развитие этих систем? И множество других.

Частично эволюция языков программирования была рассмотрена в предыдущем разделе. В данном разделе для нас, наряду с частностями, важно иметь и общее представление о процессе эволюции. Для этого важны методы исследований данных процессов. Последние опираются на принципы системных исследований, которые включают:

1. Целенаправленность – целевое назначение системы и ее способность достигать заданных целей.

2. Целостность, которая предполагает рассматривать совокупность элементов, входящих в систему как единое целое для обеспечения имержентности.

3. Иерархичность, то есть многоуровневое рассмотрение элементов системы.

4. Организованность, то есть наличие определенных структурно-функциональных связей между элементами.

5. Выявление внутренних закономерностей в системе.

6. Исследование инерциальных свойств системы, влияющих на ее дальнейшее развитие.

7. Рациональность построения системы, то есть новая система должна быть эффективнее своих предшественниц.

8. Способность к модернизации.

Для понимания направленности эволюции языков, систем и технологий программирования необходимо обратить внимание не только на проблемную их ориентацию, но и, как в материальном производстве, на развитие технологии и организации производства программ. Следуя [6], можно выделить четыре периода развития производства ПО: производство ПО для удовлетворения собственных потребностей разработчиков (до начала 60-х годов); становление товарного производства и развитие кризиса ПО (до середины 70-х годов); экстенсивное товарное производство ПО (до середины 80-х годов); становление интенсивного товарного производства.

Соответственно этим периодам развивались языки и технология программирования, начиная с программирования в кодах и заканчивая объектно-ориентированным программированием и созданием технологии адаптируемых надежных систем.

Помимо всего, это деление на этапы соответствует увеличению массовости использования ПО и быстроте темпов его модернизации.

В технологии и организации производства программ важны не только уровень абстрагирования процедур и данных, но и эффективные методы декомпозиции программ.

Начало такой декомпозиции программы положила концепция модульного программирования, предложенная в середине 50-х годов для программ, написанных на Ассемблере и Фортране. Она получила свое дальнейшее развитие в связи с появлением новых языков и концепций программирования.

Одним из таких подходов явилась концепция структурного направления [7, 8]. В современное понимание этого направления обычно включают: концепцию структурного кодирования, модульного программирования и дисциплинарного проектирования ПО. Основной целью такого подхода, которая декларировалась еще ее создателями, была разработка метода создания программ,

широко использующих средства абстрагирования с целью выделения составных частей проблемы и повышения уверенности в правильности работы конечной программы.

Для достижения этой цели предусмотрены только три вида управляющих структур: последовательность (линейный участок), выбор (без GOTO), повторение (цикл) и структура данных. Наиболее типичным представителем языков, в котором реализованы идеи структурного программирования, является язык Паскаль (Paskal) [9]. Помимо упомянутых выше управляющих структур в структуре данных, он допускает массивы, записи, файлы, наборы и классы, определяемые пользователем.

Структурное кодирование, аппарат подпрограмм и процедуры модульного программирования направлены на упорядочение в программах связей по передачам управления и накопление опыта (использование уже известных процедур). Но не менее важно уменьшить сложность в структурах данных и их связей, что наиболее просто может быть реализовано, если основываться на концепции абстрактных типов данных (АТД), одна из версий которых была реализована Виртом в языке Паскаль.

Более полно эти подходы реализованы в языке Ада [10, 11].

Концепция АТД допускает введение в язык классов объектов более общих, чем типы данных. В объектно-ориентированном программировании имеются различные модификации АТД. Так, в языке Ада достигнут новый уровень абстракции благодаря введению механизмов родовых структур, который обеспечивает возможность порождения структур – "потомков" с уточненными свойствами в процессе компиляции. Такой процесс создания потомков, называемый конкретизацией, определен для любых сегментов компиляции, в том числе для пакетов и задач.

Для целей параллельного программирования в языке Ада определены задачи, взаимодействие которых обеспечивается не только работой с общей памятью, но и механизмами синхронизации высокого уровня, называемыми "рандеву". Остановимся более подробно на ориентации языков программирования по областям применения с указанием наиболее типичных языков этого направления.

Наиболее типичны следующие области применения языков:

– вычислительные задачи (Фортран [12], Алгол [13] и др.);

– обработка деловых данных (Кобол [14]);

– моделирование (Симула, GPSS);

– символьные и алгебраические преобразования и вычисления (системы компьютерной алгебры [15–19];

– искусственный интеллект (функциональные языки Лисп [3], АПЛ [20]);

– логические языки Пролог, Мандала [3];

– многоцелевые языки (Ада [10, 11]);

– системное программирование (Си [21]);
– управление процессами (Си, Ада [36]);
– диалоговые языки (Бейсик);
– обработка списков (Лисп [3]);
– обработка строк (Снобол [22]) и другие области применения.

Эволюцию развития языков по проблемной ориентации, возможностям расширения и ориентации на компоненты языка можно проследить по таблице, заимствованной из работы [23].

Таблица

| Язык | Год | Проблемная ориентация | | | | | | Возможности расширения | Ориентация на компоненты |
|---|---|---|---|---|---|---|---|---|---|
| | | Вычислительная | Деловая | Управление процессами | Системная | Моделирование | Искусственный интеллект | | |
| FORTRAN | 1957 | + | | | | | | нет | операторы |
| ALGOL-60 | 1960 | + | | | | | | нет | операторы |
| COBOL | 1960 | | + | | | | | нет | данные |
| LISP | 1960 | | + | | | | + | ограниченные | функции и данные |
| PL/I | 1962 | + | + | | | + | | нет | операторы и данные |
| BASIC | 1965 | + | | | | + | | нет | операторы |
| SIMULA-67 | 1967 | + | | | | + | | широкие | операторы и данные |
| ALGOL-68 | 1968 | + | + | | | + | | широкие | операторы и данные |
| RTL-2 | 1969 | | | + | | + | | ограниченные | операторы и данные |
| Pascal | 1970 | + | + | | | + | | широкие | операторы и данные |
| C | 1972 | + | + | + | + | + | | – | операторы и данные |
| Ada | 1980 | + | + | + | + | + | + | широкие | операторы и данные |

Сделаем некоторые замечания к перечисленным в таблице языкам. Язык Си ориентирован на разработку системных программ, учитывающих особенности аппаратных средств. Достаточно эффективен по производительности, но обладает мало прозрачным синтаксисом и плохой обозримостью.

Симула – язык высокого уровня для имитационного моделирования. В нем впервые реализована идея языка – ядра и включает в себя с некоторыми ограничениями Алгол-60.

Отметим, что, кроме Симулы, для моделирования и при решении задач "искусственного интеллекта" (экспертные системы, дедуктивные базы данных, распознавание образов и т.д.), помимо Лиспа, используется Пролог.

Язык Ада, помимо перечисленных возможностей, обладает возможностью работы в режимах реального времени и параллельной обработки.

Отдельно хотелось бы остановиться на языке Форт [2], структура которого содержит ядро и средства, допускающие расширение, что позволяет ему быть одновременно как пользовательской, так и метасистемой, описывающей саму себя. Расширение Форта (Форт-система) позволяет создавать полный набор средств поддержки для разработки и исполнения программ; операционную систему, интерпретатор для диалогового исполнения, компилятор, ассемблер, текстовый редактор и обслуживающую программу (все написано на языке Форт). При этом требует сравнительно малых ресурсов. Все эти качества в середине 80-х годов выдвинули его по использованию на третье место после Бейсика и Паскаля в качестве средства программирования на ПК для систем обработки текстов, пакетов машинной графики, видеоигры и т.д.

Недостатком языка Форт является слабая прозрачность программ из-за недостаточно высокого уровня языка, стековой модели вычислений и т.п.

Важную роль для адаптации языка программирования к потребностям программиста играют возможности расширения языка посредством наличия средств, с помощью которых программист может создать новые объекты необходимых типов. Например, новые типы данных в языках Алгол-60, Фортран, Бейсик. В этом смысле они являются закрытыми для расширения. В языках Симула-67, Паскаль, Алгол-68 и Ада такие средства имеются, а в языке Форт их нет, но нет и ограничений на их создание и введение в язык.

Эффективность и качество всех этапов разработки и жизненного цикла программ во многом зависят от структурных возможностей языка, т.е. структуры программ и данных, управляющих конструкций. Так, Кобол допускает только одноуровневую структуру программы и все данные глобальные. В Фортране допускаются подпрограммы. В языках Алгол-60 и Паскаль используется боковая структура программ, а в языке Ада концепция модулей и родовых сегментов хорошо согласуется с концепцией структурного программирования. Структуры данных также эволюционировали от однородных агрегатных данных (массивов) в языках Алгол-60 и Фортран до иерархических агрегатов дан-

ных, например, массивы структур, содержащие подструктуру – языки ПЛ/1 [25] и Алгол-68, либо полный и "джентльменский набор", содержащий массивы, записи, множества и файлы, как в языке Паскаль. На качество разрабатываемых программ также влияют состав и вид управляющих операторов. В современных языках вместо простых конструкций операторов языков типа Фортран и Алгол-60 используются более сложные и хорошо определённые операторы типа if...the...else...end if, case...of...when...end case, for...in...while...loop...end loop.

Важным свойством современных языков является наличие средств для организации параллельного выполнения программ. В зачаточной форме эти средства наблюдаются в Алголе-60 и ПЛ/1 [29], дальнейшее развитие они получили в Симуле-67 и Алголе-68. Наибольшего развития эти средства получили в языке Ада. Отметим еще ряд некоторых особенностей языков программирования.

Отличием языкаАлгол-60 от Фортрана является строчное определение синтаксиса, это способствует единообразному пониманию правил языка как пользователями, так и разработчиками транслятора. Однако семантика языка Алгол-60 описана неформально, что может привести к неоднозначному истолкованию отдельных конструкций. Алгол-60 широко использует рекурсивность и основывается на блочности структуры программы.

Фортран имеет развитые средства описания форматов вводимых и выводимых данных как числовых, так и текстовых, удобный аппарат сегментирования больших программ, хорошее развитие функций и процедур.

По замыслу авторов ПЛ/1 (IBM) – это язык-оболочка для многочисленного решения задач обработки данных экономической информации, обработка строк и списков научно-технических задач вычислительного характера, задач управления объектами в реальном времени и задач системного программирования. Язык построен по модулярному принципу, позволяющему образовывать подмножества языка путем отбрасывания ненужных средств. Использует "интерпретацию умолчания" и в случае, если в программе имеется несколько свойств, но программист не указал конкретного, то автоматически будет выбрано свойство, наиболее вероятное в данном контексте.

Средства ПЛ/1 позволяют программно организовать параллельную мультипроцессорную обработку за счет указания ветвей программы, которые могут выполняться параллельно.

Алгол-68 – типичный пример языка – ядра, на базе которого могут создаваться новые языки путем конструирования новых средств из имеющихся элементов языка - ядра. Введено понятие вид, относящееся не к переменному, обозначающему объект, а к значению, приписываемому объекту. Множество видов в целом неограничено, что позволяет вводить свои операции, описав их.

Один из подходов адаптации к внутренним (машинным особенностей) и внешним (требования задач и программиста) условиям языка программирования основан на создании универсальных языков программирования. Это приводит к необходимости объединения существенных черт и специфических средств современных машинно- и процедурно-ориентированных языков. Такое объединение может быть осуществлено на основе концепции язык – ядро, язык-оболочка. Концепция язык – ядро состоит в наборе средств, на основе которых можно конструировать процедурно-ориентированный язык для конкретного применения. Примером такого подхода является Алгол-68. Концепция языка – оболочка представляет конгломерат различных средств, известных процедурно-ориентированных и машинно-ориентированных языков. Разные подмножества языка – оболочки могут использоваться в качестве самостоятельных процедурно-ориентированных средств.

На наш взгляд, более рационально использовать ядро и средства расширения, как это делается в Симуле, но одновременно использовать и идеологию оболочки для подстройки и адаптации языка к внутренним и внешним условиям применения, включая и транспортабельность. Значительным шагом в развитии языков и систем программирования явились структурный подход и абстрактные типы данных. Поэтому в следующем разделе остановимся на этих подходах более подробно.

### 11.4. Структурное программирование и абстрактные типы данных

Структурный подход в программировании [7, 8] включает в свой состав следующие три части: нисходящая разработка, структурное программирование и сквозной структурный контроль. При этом нисходящая разработка программ предполагает, что программирование и проектирование ведутся сверху вниз. Помимо этого, используется идея уровней абстракции, которые реализуются в создаваемой программе уровнями программных модулей. Для понимания идеи абстрагирования можно воспользоваться определением Фроста [8], который понимает абстрагирование как процесс "обобщения, при котором внимание концентрируется на сходстве явления и предметов, и они объединяются в группы на основе этого сходства, давая тем самым нужную абстракцию".

В структурном подходе важную роль играют модульность и вертикальное управление. В широком смысле, структурный подход, как уже отмечалось выше, предполагает широкое использование идеи абстрагирования с целью выделения составных частей проблемы и повышения уверенности в правильности конечной программы за счет доказательства правильности программ.

В узком смысле, структурный подход состоит в том, что для передачи управления в программе используют только три конструкции: последовательную (линейный участок), условную (если ... то ... иначе) и итеративную пере-

дачу управления, а также ограниченное использование глобальных переменных. Результатом такого подхода является то, что каждая сложная программа разбивается на модули, имеющие только по одной точке выхода, и имеет древовидную структуру. Такой подход облегчает восприятие и отладку программ, а также позволяет разработку системы многими программистами. Идея абстрагирования, возникшая в конце 70-х годов прошлого века, тесно связана с рассмотрением абстрактных типов данных (АТД).

Следуя [35], АТД – принцип определения типа данных через операции, которые могут исполняться над объектами данного типа. При этом вводится следующее ограничение: значение таких объектов может модифицироваться и наблюдаться только путём использования этих операций. Описание типа данных через имеющиеся операции предоставляет всю необходимую для использования этого типа данных информацию и обеспечивает максимальную свободу реализации. Помимо этого, появляется возможность создания "библиотеки" полезных АТД – списков, очередей и т.п.

Типичной реализующей АТД в программе является реализация с помощью многопроцедурного модуля, содержащего локальные данные для представления значения данного типа, и каждая процедура реализует одну из операций, связанную с конкретным типом. При этом доступ к локальным данным модуля может осуществляться только со стороны данных процедур. Важно отметить, что принципы АТД должны быть заложены в самом языке программирования. Первым языком, способным работать с АТД, стал язык Симула, в котором была реализация концепции класса. В языке Модула реализовано АТД класс, а в языке Ада – пакет.

Строго говоря, АТД представляет собой тройку (D, F, F), состоящую из множества: областей D, функций F, каждая из которых существует и изменяется в D, и аксиом A, которые задают свойства функции в F. В последнее время вместо требования наличия аксиом A выдвигается требование, что все свойства объектов АТД определяются семантикой его основных операций.

Появление АТД в полной мере соответствует принципу сокрытия информации, который является одним из атрибутов эволюции языков программирования. Последовательность развития этого принципа следующая: абстракция данных, позволяющая использовать данные, не зная их организации и создания; абстракция действий в виде процедур; абстракция типа данных и, наконец, абстракция объектов в объектно-ориентированном программировании. Последний аспект более подробно рассмотрен позже. Есть два аспекта структур данных – это структуры, отвечающие модели решаемой задачи, и отображение собственно абстрактных структур данных в абстрактные структуры памяти. В соответствии с [27] к первому аспекту относятся: строки, массивы, очереди и стеки, таблицы, деревья и ориентированные графы, а ко второму – векторы и списки.

При этом, как отмечает Д.П. Шишков [28]: "Математики изучают конечные и бесконечные объекты, знаковые образы которых состояли из небольшого количества знаков с богатой семантикой. Наоборот, в компьютерной информации компьютерные системы обрабатывают многочисленные данные, но с бедной семантикой... данные должны быть организованы (структурированы, упорядочены) в компьютерной памяти для того, чтобы их можно было искать эффективно. В работе [28] выделим следующие четыре вида абстрактных структур для и от данных. Абстрактные структуры для данных включают: абстрактные структуры памяти (АСП), где принципиально важной структурой является вектор, который наиболее адекватно отображает память, сеть и обобщенный вектор.

Собственно абстрактные структуры, которые делятся на вырожденные (константы и переменные) и невырожденные структуры. Невырожденные структуры делятся на таблицы, линейные структуры (списки), древовидные структуры, сетевые структуры без ориентированного цикла, схемы и т.д.

Таблицы делятся на неупорядоченные, упорядоченные и индексно-последовательные, которые составляют класс динамических таблиц. Класс статических таблиц включает в свой состав логические записи и массивы.

Структуры с порядком включают линейные структуры вида списки (строки), стеки, очереди, деки, а также нелинейные структуры вида древовидных (связных и несвязных) и сетевые структуры без ориентированного цикла.

В работе [28] приводится классификация собственно абстрактных структур данных. Помимо этого, рассматривается математический тип данных, содержащий следующие семь множеств: алфавит из букв (знаков), порождающих операции (первичные и комбинированные конструкторы данных (множество слов определенного формата над заданным алфавитом)); форматы (множество правил и отношений), операции (одно- и многоаргументные функции, анализаторы, многоаргументные операции), аксиомы (множество аксиом, описывающих отношения между элементами данных).

В полном объеме положения АТД не реализованы ни в одном из существующих языков программирования. Наиболее полно всем требованиям АТД, кроме модульности, удовлетворяют только приватные (личные) типы языка Ада. Встроенные типы языков Ада и Паскаль не являются АТД в полном понимании основных требований. Как мы уже отмечали выше, пакеты языка Ада удовлетворяют требованию модульности, но для них не выполняется то требование АТД, когда тип определяет класс объектов (значений), которые могут принимать выражения или переменные, так как пакет описывает только конкретный объект, а не класс объектов.

В общем, концепция языка Ада отражает многие достижения методологии структурного программирования, направлена на систематическое использование абстракции для управления массой деталей. Это помогает проектиро-

вать достаточно прозрачную программу [7]. Так, язык Ада содержит определяемые типы, пакеты, задачи, родовые сегменты, рандеву и др., что отличает его от языков ПЛ/1, Фортран, Алгол-60, Кобол и т.д.

### 11. 5. Дальнейшие пути развития языков и систем программирования

Аппарат подпрограмм, процедур, модульное и структурное программирование соответствуют определенным этапам технологии программирования и во многом направлены на декомпозицию программ, накопление опыта и обеспечение упорядочения связей по передачам управления в программах. Однако это еще не является решением всей проблемы в целом и, в частности, это касается структур данных и их связей. Наибольшее влияние на решение данной проблемы оказали теория и практика АТД, рассмотренные в предыдущем разделе.

Помимо этого, большую роль будут играть динамичность средств декомпозиций и композиций программ. В этой связи можно согласиться с высказыванием В.Н. Редько [18]: "Программирование, как и решение любых других задач, сопряжено с использованием системных стратегий – чередованием декомпозиций и композиций объектов (задач, программ, данных и т.п.)".

Дальнейшим развитием систем программирования явилось использование объектно-ориентированного проектирования [30], которое позволяет программировать в привычных для прикладного программиста понятиях и отношениях конкретной предметной области. В частности, такие средства имеются в языковом инструментарии SQL [31]. Объектно-ориентированная методология направлена на обеспечение процессов анализа, проектирования, программирования и описания процесса разработки ПО в целом.

В объектно-ориентированном программировании во главу угла ставится понятие объекта, рассматриваемое на высоком уровне абстракции структуры данных, функций, композиций и декомпозиций.

Сутью объектно-ориентированного подхода является декомпозиция программируемой задачи на объекты. При этом объекты – общее название для процессов, явлений, понятий, сущностей и т.п., относящихся к предметным областям. В концептуальных моделях объекты объединяются в типы объектов, описываемые одним общим для каждого набора атрибутом. При этом экземпляры объектов, как и элементы типа, уникальным образом идентифицируются посредством ключа типа объекта. Под сущностью в объектно-ориентированном программировании понимается элемент модели предметной области, означающий объект, предмет, понятие и т.д. Система программ представляет совокупность активных объектов, взаимодействующих

путем посылки сообщений. Объект имеет два аспекта – функции (оператора) и данных (операндов). Последовательность выполняемых функций (операторов) определяется динамически в процессе выполнения программы. Допускается группирование объектов в классы и последовательности, являющиеся инструментом абстракции, дающие возможность наращивать знания о предметной области.

Одним из наиболее типичных для реализации этой методологии является язык UML [30, 32]. Для этого языка характерно использование таких средств, как пакет, модель, точка зрения, подсистема. Помимо этого, содержатся диаграммы классов объектов, использования, последовательности, сотрудничества, состояний, действий, реализаций и т.п.

Очень важной особенностью объектно-ориентированной методологии, которая сделала шаг вперед по сравнению с предыдущими статичными подходами, есть ее динамичность.

Помимо этого, в объектном программировании допускается разбиение множества основных операций на собственные операции объектов, так как операции считаются составной частью объектов, и операции общие для всех объектов данного типа.

В настоящее время объектно-ориентированный подход является одним из наиболее прогрессивных. Но и он имеет по крайней мере два существенных недостатка: достаточно сложную методологию и организацию. Так, язык UML содержит более 100 различных обозначений и т.п. Помимо этого, реализация программ на язык UML требует больших временных затрат и других ресурсов.

Помимо развития структур программ и данных, происходит совершенствование выразительных средств языков и систем программирования. Более подробно с этой проблемой можно ознакомиться в работе Б. Хигмана [33], где прослеживается эволюция развития языков программирования, их преемственность и последовательное усложнение лингвистических особенностей, включая появление принципиально новых выразительных средств. Подобные сравнения сделаны также в работах [3, 15, 33, 36].

Среди этих выразительных средств, имеющих принципиально важное значение для развития языков и систем программирования, по нашему мнению, имеют следующие два направления: средства систем компьютерной алгебры [15–19, 37–41] и работа со знаниями [3, 53, 42, 43]. Первое направление представляет интерес в связи с тем, что позволяет вести аналитические и численно-аналитические (символьные и алгебраические) вычисления и имеет большое разнообразие представления математических объектов, что требует наличия мощных средств распознавания, приведения подобных и т.д. А второе направление интересно с точки зрения работы со знаниями как на основе языков Лисп, Пролог, Смолток, Лого и др., так и при использовании нейронных сетей, кото-

рые способны к обучению и выступают в виде структуры ассоциативной памяти.

Второе и частично первое направления используются для доказательства и логического программирования, программирования игр, моделирования, обработки сигналов и распознавания образов, решения задач машинного зрения и обработки изображений, робототехники и автоматизации производства, машинного проектирования и т.д.

Сегодня используют основные методы представления знаний, такие, как фреймы, семантические сети, продукции, различные логики и т.д. Основными методами программирования для их реализации являются функциональное, объектное, логическое и т.д. программирование.

Основные типы знаний, соответственно классификации [3], следующие:

1. Объекты и их свойства. Объект – это существующие в прикладной области универсальные понятия и их представители.

2. События. События описывают участие объектов в деятельности и ситуациях.

3. Действия. Действия в интеллектуальной деятельности, представляющие процедурные знания о том, каким образом что-то делается. Например, каким образом из старых данных на основе правил выводятся новые.

4. Метазнания. Метазнания – это знания о знаниях и их использование.

Относительно первого направления наиболее перспективным в части распознавания объектов языка, механизмов приведения, АТД и других механизмов на сегодняшний день представляется язык Аналитик-2000.

В современных универсальных системах, таких, как AXIOM, MAPLE, MATHEMATICA и других, над каждым классом математических объектов, например, полиномы, векторы, матрицы с символьными элементами, тензоры и т.д., определены свои наборы операции и функции. Эти системы велики по объему и состоят из многих специализированных пакетов. Во многих таких системах нет четко выделенного ядра и они достаточно сложны для обучения.

Важными направлениями развития будущих языков, как уже отмечалось раньше, является создание механизмов адаптации языков и систем программирования к внутренним и внешним условиям применения, включая приспосабливающихся (обучающихся), самоопределяющихся, позволяющих писать компилятор на самом этом языке; расширение понятия абстрактных типов данных и структур; использование концепции "расширяющейся вселенной", включающей концепцию "расширяющегося ядра" с набором конструктивных элементов для возможности создания индивидуально-корпоративных версий; средства контроля правильности функционирования и многие другие.

Будущие программные языки и системы в той или иной степени должны учитывать разработанную П.С. Сапатым технологию интеграции распределенных и динамических систем, основанную на языке WAVE (ВОЛНА) [44]. П.С.

Сапатым разработана новая идеология, методология и технология интеграции и координации больших открытых распределенных и динамических систем, которые могут объединять компьютеры, компьютерные сети, всевозможные пилотируемые и беспилотные платформы (воздушные, наземные, над- и подводные, космические), автономные мобильные роботы, а также мобильные компьютеры.

Важной особенностью такого подхода является возможность создания и моделирования надежных распределенных систем с высоким уровнем отказоустойчивости. Язык WAVE является языком системного программирования и управления, позволяющим большую часть функций синхронизации, координации, обмена командами, данными и физическими объектами переместить на уровень интерпретации. Это позволяет программировать общесистемное поведение на смысловом уровне, центрируя внимание программиста на вопросах системной целостности и выполнении глобальных задач. Некоторые черты такого подхода реализованы в языке Java.

Роль сетевого взаимодействия в современной жизни и в будущем рассмотрена в работе [4]. Там же показано, что вычислительные средства подчинены закону обобщенной симметрии по отношению к человеку. Из этого следует, что следующей фазой развития языков и систем программирования будет семантическая, но, учитывая принцип смешанного экстремума, который является одним из законов эволюции, эти языки и системы будут иметь смешанный характер.

Подтверждение необходимости мобильности и адаптации в эволюционном развитии вычислительных средств показано в работах [4, 5], что естественным образом переносится на языки и системы программирования.

Агентная технология программирования предполагает, что агенты характеризуются следующими свойствами: автономностью, индивидуальным представлением мира, способностью к коммуникации и кооперации, интеллектуальным поведением, наличием собственной внутренней архитектуры.

Агенты могут мобильно перемещаться по сети и из распределенных ресурсов собирать информацию, обрабатывать и выполнять ее. Помимо этого, агенты могут быть взаимодействующие (совещательные) и реактивные.

В зависимости от присущих агенту атрибутов различают следующие типы: взаимодействующий, интерфейсный, взаимодействующий обучающий и смарт агент [49].

Наш обзор языков был бы неполным, если вспомнить активно развивающееся в последнее время агентная технология программирования, основанная на сетевом взаимодействии, и генное программирование, используемое для моделирования в биологии.

И завершим обзор, получивший в последнее время распространение, технологией генетического программирования. Генетическое программирование –

технология которого основывается на том, что программы способны эволюционировать. При этом компьютер генерирует за инструкцией случайные программы, постоянно их изменяя, и выбирает из них те, которые справляются с решением поставленной задачи наилучшим образом. Одним из пионеров этого направления является Джон Коза – ученый-компьютерщик из Стенфордского университета.

Известно, что будущее рождается в прошлом. В связи с этим многие элементы будущих языков и систем имеются в ныне существующих языках и системах. Именно поэтому пришлось анализировать наиболее характерные языки и эволюцию средств их развития.

**Выводы**

На основе проведенного в настоящей работе анализа языков и систем программирования, а также учитывая анализ эволюционного развития вычислительных средств [2, 4, 5, 45, 46] и, в частности, принципа смешанного экстремума и обобщенного закона зеркальной симметрии, можно прогнозировать следующие особенности языков и систем программирования:

1. Языки и системы программирования будут объединять в себе различные ориентации (системную и прикладную, символьные и вычислительные, императивные и декларативные и т.д.).

2. Получит дальнейшее развитие абстрагирование данных, знаний, структур и средств управления.

3. Языки будут иметь ядро и специализированные средства для адаптации и саморазвития, позволяющие объединять различные специализированные и проблемно-ориентированные подсистемы языка, а также создавать свои персонифицированные версии.

4. На смену ныне широко используемым структурно и объектно-ориентированным технологиям придет семантическое (смысловое) программирование, использующее в той или иной мере существующие технологии.

5. Будущие языки и системы программирования будут содержать механизмы обеспечения высокого уровня отказоустойчивости, гарантоспособности и прозрачности создаваемых программных продуктов.

6. Должны обеспечивать накопление полученных данных и знаний.

7. Наряду с языками, ориентированными на стационарные компьютерные платформы, будут существовать их версии, ориентированные на мобильные компьютерные платформы со средствами сетевого взаимодействия.

8. Иметь средства поддержки параллельной и параллельно-последовательной работы.

9. Иметь средства поддержки особенностей используемой компьютерной платформы.

10. Должны соответствовать соответствующим стандартам на унификацию и переносимость программ.

## Список литературы


1.*Теслер Г.С.* Место и роль алгоритмического базиса в решении проблемы производительности // Математические машины и системы. – 1997. – № 1. – С. 25 – 33.

2.*Теслер Г.С.* Принцип смешанного экстремума как основа эволюционного развития вычислительных средств // Математичні машини і системи. – 2002. – № 1. – С. 3 –13.

3. *Хювенен Э., Сеппенен Й.* Мир Лиспа: Пер. с финск. – М.: Мир, 1990. – Т.1: Введение 199а – в язык Лисп и функциональное программирование. – 447 с.

4. *Теслер Г.С.* Перспективы развития вычислительных средств с сетевым взаимодействием // Математичні машини і системи. – 2001. – № 1, 2. – С. 3 – 11.

5. *Теслер Г.С.* Сопоставление эволюции развития вычислительных средств и растительного мира // Математичні машини і системи. – 2002. – № 3. – С. 155 –166.

6. *Марков А.С., Милов М.П., Пеледов Г.В.* Программное обеспечение ЭВМ. – М.: Высшая школа, 1990. – 127 с.

7. *Дал У., Дейстра Э., Хоар К.* Структурное программирование: Пер. с англ. – М.: Мир, 1975. – 247 с.

8. *Хьюз Дж., Мигтом Дж*. Структурный подход к программированию: Пер. с англ. – М.: Мир, 1980. – 282 с.

9. *Грогоно П.* Программирование языка Паскаль: Пер.с англ. – М.: Мир, 1982. – 384 с.

10. *Вегнер П.* Программирование на языке Ада: Пер. с англ. – М.: МИР, 1983. – 240 с.

11. *Пайл Я.* АДА – язык встроенных систем: Пер. с англ. – М.: Финансы и статистика, 1984. – 238 с.

12. *Салтыков А.И., Макаренко Г.И.* Программирование на языке Фортран. – М.: Наука, 1976. – 55 с.

13. *Халилов А.И., Ющенко А.А.* Алгол-60 / Под ред. проф. Е.Л. Ющенко. – Киев: Вища школа, 1975. – 352 с.

14. *КОБОЛ*: Учебное пособие / Под ред. Е.Л. Ющенко, Л.П. Бабенко, Е.И. Машбица. – Киев: Вища школа, 1973. – 292 с.

15. *Осипов Л.А.* Язык аналитики, его сравнение с языком Алгол и Фортран. – М.: Наука, 1982. – 162 с.

16. *Клименко В.П., Ляхов А.Л., Фишман Ю.С.* Основные тенденции развития языков систем компьютерной алгебры // Математичні машини і системи. – 2002. – № 2. – С.168 – 175.

17. *Попов Б.О.* Розв`язування математичних задач у системі комп`ютерної алгебри Maple. – Київ: VIP, 2001. – 312 с.

18. *АНАЛИТИК* – 2000 / А.А. Морозов, В.П. Клименко, Ю.С. Фишман и др. // Математичні машини і системи. – 2001. – № 1,2. – С. 66 –99.

19. *Аналитик.* Численно-аналитическое решение задач на малых ЭВМ / Б.А. Бублик, В.П. Клименко, С.Б. Погребинский, Ю.С. Фишман. – Киев: Наукова думка, 1987. – 143 с.

20. *Гильман Л., Роуз А.* Курс АПЛ: диалоговый подход: Пер. с англ. – М.: Мир,1979. – 52 с.

21. *Keringhan B.W., Ritchie D.M.* The C Programming Lranguange. – Englewood Cliffs: Prentice. – Hall, 1978.

22. *Грисуолд Р., Поудн Дм., Полонски И.* Язык программирования Снобол–4: Пер. с англ. – М.: Мир, 1980. – 268 с.

23. *Кравец В.А., Шпальберг А.Я.* Зарубежные ЭВМ. Образование и программное обеспечение. – Харьков: Основа при Харьковском университете, 1991. – 216 с.

24. *Баранов С.Н., Наздронов Н.Г.* Язык Фортран и его реализация. – Ленинград: Машиностроение, 1988. – 157 с.

25. *Гнеденко Б.В., Королюк В.С., Ющенко Е.Л.* Элементы программирования. – М.: Физматгиз, 1963. – 348 с.

26.*Толковый* словарь по вычислительным системам / Под ред. В. Илингуорта, Э.Л. Глейзера, И.К. Пайла: Пер.с англ. – М.: Машиностроение, 1991. – 560 с.

27. *Лебедев В.Н.* Введение в системы программирования. – М.: Статистика, 1975. – 309 с.

28. *Шишков Д.П.* Абстрактные структуры: для и от данных // Математичні машини і системи. – 2000. – № 2, 3. – С. 23 – 317.

29. *Безбородов Ю.М.* От Фортрана - к PL/1. Основы языка PL/1. – М.: Наука, 1984. – 208 с.

30. *Буч Г.* Объектно-ориентированное проектирование с примерами применения: Пер. с англ. – Киев: Диалектика, 1992. – 519 с.

31. *Фути К., Судзуки Н.* Языки программирования и схемотехника СБИС: Пер. с японск. – М.: Мир, 1988.

32. *Фаулер М., Скотт К.* UML. Основы. – М.: Символ – Плюс, 2001. – 192 с.

33. *Хигман Б.* Сравнительное изучение языков программирования: Пер. с англ. – М.: Мир, 1974. – 204 с.

34. *Безбородов Ю.М.* Сравнительный курс языка PL/1 (на основе Алгола-60). – М.: Наука, 1980. –191 с.

35. *Цейтлин Г.Е.* Введение в алгоритмику. – Киев: Фарт, 1998. – 310 с.

36. *Языки* программирования Ада, Си, Паскаль: Сравнения и оценка: Пер. с англ. / Под ред. А.Фьюэра, Н.Джехани. – М.: Радио и связь, 1989. – 360 с.

37. *Дэвеннорт Дж., Сирэ И., Турнье Э.* Компьютерная алгебра. – М.: Мир, 1991. – 352 с.

38. *Потемкин В.Г.* Система MATLAB: Справочное пособие. – М.: Диалог – МИФИ, 1998. – 350 с.

39. *MATHCAD* 6.0 PLUS. Финансовые, инженерные и научные расчеты в среде Windows 95: Пер. с англ. – М.: Информационно-издательский дом "Филин", 1996. – 712 с.

40. *Капустина Т.В.* Компьютерная система MATHEMATICA 3.0. – М.: СОЛОН, 1999. – 240 с.

41. *Еднерал В.Ф., Крюков А.Р., Родионов А.Я.* Язык аналитических вычислений REDUCE. – М.: Из-во МГУ, 1989. – 177 с.

42. *Морозов А.А., Ященко В.А.* Интеллектуализация ЭВМ на базе нового класса нейроподобных сетей. – Киев: Тираж, 1997. –126 с.

43. *Нейронные* сети в системах автоматизации / В.И. Архангельский, И.Н. Богаенко, Г.Г. Грабовский, Н.А. Рюмшин. – Киев: Техника, 1999. – 364 с.

44. *Sapaty P.S.* Mobil Processing in Distributed and Open Environments. – New York: Yohn Willy, 1999. – 436 p.

45. *Эккель Б.* Философия Java. Библиотека программиста: Пер. с англ. – СПб: Питер, 2000. – 880 с.

46. *Теслер Г.С.* Концепция создания вычислительных средств с высоким уровнем отказоустойчивости // Математичні машини і системи. – 2002. – № 2. – С. 176 –183.

47. *Теслер Г.С.* Информация – феномен природы: Роль информации в естественной и искусственной природе // Математичні машини і системи. – 2003. – № 1. – С. 152 – 165.

48. *Редько В.Н.* Программология: прошлое, настоящее, будущее // Вестник Международного Соломонова университета. – 1999. – № 1. – С. 24 – 63.

49. *Sowtware* Agents: An Overview // VRL: http: // www. labs / publich / papers / rewviw2. htm # agent.