СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНСТИТУТ ИНФОРМАТИКИ И ТЕЛЕКОМУННИКАЦИЙ

**Реферат**

Разнообразие языков программирования и сред разработки. Краткий экскурс

Выполнил:

студент 1 курса

группы БПИ24-02

Гордов В.Т.

Проверил:

Красноярск, 2024

## Введение

В современном мире информационные и компьютерные технологии играют очень важную роль в развитии общества. Они уже давно глубоко проникли во все сферы нашей жизни и этот процесс не прекращается и по сей день. Создание программного обеспечения основано на использовании различных языков программирования и сред разработки, которые имеют различные преимущества и недостатки и различаются по сферам применения.

Актуальность темы реферата обуславливается тем, что разнообразие языков программирования и сред разработки постоянно растёт – одни набирают популярность, вторые – теряют свои позиции, а третье – находятся в процессе разработки. Для того, чтобы лучше понимать, какой инструментарий разработки следует использовать под определённые задачи, нужно иметь представление обо всём разнообразии этих инструментов и тенденциях их развития.

Задачи работы:

1. Обозреть историю развития языков программирования
2. Ознакомиться с методами классификации языков программирования
3. Рассмотреть краткую историю развития сред разработки
4. Рассмотреть инструменты, используемые различными средами разработки

Языки программирования

Язык программирования – это формальный язык, предназначенный для записи компьютерных программ, определённый набором семантических, лексических и синтаксических правил, которые определяют процесс написания программы пользователем и её дальнейшее исполнение компьютером.

## Краткая история языков программирования

В 1945 году венгеро-американский математик Джон Фон Нейман разработал две важные концепции:

Первая – «метод совместного использования программ»: аппаратное обеспечение компьютера должно быть простым и не требовать ручной настройки для каждой отдельной программы. Наоборот, необходимо использовать сложные конструкции для управления простым оборудованием, чтобы облегчить и ускорить процесс программирования.

Вторая – «условная передача управления»: код должен представлять собой не чётко определённый набор шагов программы, а, во-первых, набор блоков кода и подпрограмм, к которым можно переходить в любом порядке, а во-вторых – иметь возможность разветвляться на основе циклов (for, while) и логических операторов (if, else).

В 1949 году американской компанией EMCC был создан язык «Short Code». Это был первый компьютерный язык для ЭВМ, операторы в котором кодировались двухсимвольными сочетаниями – единицы и нули. В 1951 году Грейс Хоппер написала для него компилятор под названием «A-0». Short Code стал первым шагом для появления более сложных языков программирования, он считается языком первого поколения.

В дальнейшие несколько лет появляются языки второго поколения, известные как «языки ассемблера». В них появились мнемоники – символьные обозначения машинных команд, а также возможности сопоставления имён с адресами в памяти компьютера, что делает эти языки более понятными для понимания человеком по сравнению с языками первого поколения.

В 1957 году появляется язык «Fortran», название которого расшифровывается как «FORmula TRANslating system». Он был разработан для научных вычислений, поэтому хорошо справлялся с числами, но не с другими типами задач. В 1958 году создаётся язык «Lisp», используемый для работ в области искусственного интеллекта, а в 1959 году разработан язык «Cobol», который позиционировался как язык для бизнес-вычислений. В этих языках появляются такие концепции, как списки, автоматическое управление памятью, «вынос мусора» и другие, а также они могут исполняться на любой ЭВМ, на которой имеется транслятор для языка. Это делает данные языки более абстрактными и универсальными и причисляет их к третьему поколению.

1958 год – создан язык «Algol», обладавший новыми концепциями, например, рекурсивным вызовом функций и формальной грамматикой. Основной его вклад заключается в том, что он привёл к появлению таких языков, как Pascal, C и Java.

В 1968 году Никлаус Вирт создаёт язык «Pascal». Он сочетал в себе лучшие особенности языков, использовавшиеся в популярных языках того времени, таких как Fortran, Algol и Cobol. В Pascal появилось сочетание функций ввода/вывода и математических функций, что быстро дало ему популярность, однако в языке не были реализованы динамические массивы и группы переменных, из-за чего его актуальность упала в последствии.

На смену Pascal в 1972 году приходит «C», разработанный Деннисом Ритчи. Он обладал большой скоростью и мощностью, что компенсировалось трудностью чтения и восприятия кода. Тем не менее, в нём были устранены многие недочёты менее совершенного Pascal, благодаря чему он достаточно скоро обрёл широкую популярность у пользователей. В C были такие возможности, как динамические переменные, многозадачность, обработка прерываний, разветвление и сильный низкоуровневый ввод-вывод. Этот язык считается переходным между первыми низкоуровневыми и современными высокоуровневыми.

На стыке 70-х и 80-х годов начинает разрабатываться новый метод программирования – объектно-ориентированное программирование (ООП). Его основной принцип заключался в том, что при написании кода использовались объекты – фрагменты данных, обладающие определённым набором свойств и функций. В 1983 году Бьёрн Страуструп, сотрудник компании Bell Labs, создаёт язык «C++», который являлся модифицированной версией C, поддерживающей ООП.

Начиная с 80-х годов наступает так называемый «период консолидации» - определённые языки становятся основными в своей области. Так, C++ объединяет черты ООП и системного программирования, Ada – используется правительством США для разработки военных систем, ML и Lisp принимаются в качестве стандартов функционального программирования. В этот период основной упор шёл не столько на разработку новых парадигм программирования, но на использование уже существующих для дальнейшего развития в сфере.

Важная тенденция того времени – применение модулей в программировании. Модуль – это объёмная единица организации кода. В последствии этот элемент стал внедряться во многие языки.

В 90-х годах с появлением и активным развитием Интернета получили распространение языки веб-программирования: Java, JavaScript Perl, PHP, Ruby и другие. Они активно развиваются и используются и по сей день. Также в 1991 году появляется «Python» - высокоуровневый язык, делающий акцент на производительность разработчика и удобство написания кода, что делает его одним из самых популярных языков на текущий момент.

Начиная с 2000-х годов языки начинают интегрировать с базами данных, появляются языки с открытым исходным кодом. Появляются «C#», «Scala», «Go» и многие другие.

В 2010-х годах многие корпорации начинают разрабатывать свои языки под собственные нужны. Так, «Rust» - Mozilla Research, «Dart» - Google, «Swift» - Apple, а также многие другие.

На сегодняшний день существует более восьми тысяч языков программирования, когда-либо придуманных человечеством и каждый год их число увеличивается.

## Классификация языков программирования

Общепринятой системы таксономии языков программирования не существует. Есть множество различных черт, по котором можно провести классификацию, например, одни из них разделяют языки по техническим признакам, другие – по доминирующим чертам, третьи – сопровождаются обилием заблуждений и субъективных мнений, но тем не менее, широко распространены на практике.

Рассмотрим основные черты, по которым принято классифицировать языки программирования:

1. **Высокоуровневые и низкоуровневые языки**:

Формально не существует такого критерия языка как «уровень». Условно этот термин имеет два значения:

1. «Приближённость» языка к человеческому языку и образу мысли.
2. «Отдалённость» от семантики машинного кода.

Эти два понятия не обязательно идут бок о бок, как кажется на первый взгляд – существуют языки, которые традиционно считались «высокоуровневыми», но реализовывались аппаратно (Lisp-машина, Java Optimized Processor), и наоборот.

По степени высокоуровневости языки принято делить на пять поколений:

1. Машинные языки – языки, реализуемые непосредственно на машинном уровне – программы пишутся в двоичном коде.
2. Языки ассемблера – в самых простейших реализована мнемоника над машинным языком, позволяющая записывать команды и их параметры в память. При этом многие языки ассемблера включают в себя макроязык – возможность записывать и использовать макросы. Программы на таких языках пишутся под конкретные устройства – перенос с одной машины на другую, зачастую, невозможен.
3. Третье поколение – языки, получившее наибольшее распространение. Появление и развитие более высокоуровневых языков в 1970-х годах стало ответной реакций на предельное повышение сложности программ, из-за которого программистам становилось всё труднее и труднее работать с ними, что в конечном счёте привело к убыткам и застою в индустрии. Одновременно с языками третьего поколения стали появляться различные стили и подходы к программированию, используемые в конкретных условиях для конкретных задач – парадигмы.
4. Четвёртое поколение – сюда относятся языки запросов, языки опций и параметров, генераторы приложений, комбинированные пакеты баз данных. Это не более ранние универсальные языки, а узкоспециализированные языки, проблемно-ориентированные на использование в конкретной области. Примеры: Empress, UNIFACE, NOMAD.
5. Пятое поколение – эта категория не является общепринятой, чаще употребляют термин «языки сверхвысокого уровня» (англ. very high level language). Чаще всего сюда относят логические языки (основанные на математической логике). Это языки, реализация которых включает существенную алгоритмическую составляющую – даже небольшой фрагмент исходного кода требует сложных вычислений.

В большинстве случаев высокоуровневые языки порождают машинный код большего размера и исполняются медленнее низкоуровневых языков. Тем не менее, языки высокого уровня позволяют программисту тратить меньше времени на сам процесс написания кода и сосредоточиться на других задачах, из-за чего конечная эффективность написанной программы может повышаться. Зачастую, скорость разработки, трудоёмкость модификации и другие показатели программы оказываются гораздо важнее, чем непосредственно скорость исполнения программы. Поэтому в последние десятилетия эволюция языков программирования движется в сторону поддержки и развития высокоуровневых языков и использования методов оптимизации кода – в общем смысле экономически более целесообразно исполнение большего числа конструкций в единицу времени.

1. **Способы реализации языка**:

Выделяется три принципиально разных способа реализации языка программирования: компиляция, интерпретация и встраивание.

1. Компиляция – исходный код программы сначала преобразуется в машинный код с помощью специальной программы – компилятора. В результате получается исполнимый модель, который может быть запущен как отдельная программа.
2. Интерпретация – операторы программы выполняются сразу, друг за другом с помощью специальной программы – интерпретатора. Таким образом реализуются конструкции, допускающие динамические изменения на этапе выполнения (модификация и создание новых подпрограмм).
3. Встраивание – такие языки реализуются зависимым образом – то есть «внутри» другого языка. Они не имеют целостной символьной системы и не обладают Тьюринг-полнотой. Также их называют «языками, реализованными поверх или на основе данного языка».

Существует заблуждение, согласно которому способ реализации языка является присущим конкретному языку свойством. В действительности это не так – один и тот же язык может выступать как компилируемый в одних случаях, и как интерпретируемый – в других. Поэтому разделение языков на способы реализации весьма условно, что делает их строгую классификацию по этому признаку невозможным.

Компиляция и интерпретация часто противопоставляются друг другу. Компилируемые языки исполняются быстрее (иногда в десятки и сотни раз), чем интерпретируемые, но при этом программа должна быть скомпилирована при каждом новом изменении, что делает поэтапное отслеживание процесса работы программы, а также модификацию программы прямо во время исполнения невозможным – в отличие от интерпретируемых языков.

1. **Парадигмы программирования:**

Парадигма в программировании – это совокупность идей и понятий, определяющих стиль написания компьютерных программ, то есть организацию вычислений и структурирование работ, выполняемых компьютером. Практические все современные языки позволяют использовать различные парадигмы при работе – такие языки называются мультипарадигменными. Помимо этого, существующие парадигмы по смыслу могут пересекаться друг с другом – одни и те же явления и определения могут описываться совершенно по-разному.

Основные выделяемые парадигмы:

1. Императивная – код состоит из последовательно исполняемых команд, при этом полученные при выполнении этих команд данные могут записываться в память и читаться из памяти последующими инструкциями. Активно используются операторы присваивания, именования переменных, а также подпрограммы. Противоположность декларативного подхода.
2. Декларативная – в этой парадигме описывается ожидаемый результат решения задачи, а не способ его получения. Здесь не используется понятия состояния (переменные и операторы присваивания), архитектура в императивном понимании здесь отсутствует. Противоположность императивного подхода.
3. Функциональная – подвид декларативного подхода. Функции здесь рассматриваются не как подпрограммы (в отличие от императивного подхода), а в своём математическом смысле. В императивном программировании результат функции может изменяться в зависимости не только от входных переменных, но и от внешних. В функциональном подходе это исключено – функция будет выдавать одинаковые результаты при одинаковых входных данных.
4. Объектно-ориентированное программирование – при таком подходе программа рассматривается как набор объектов, взаимодействующих друг с другом, при этом каждому объекту задаются определённые свойства и поведение. Использование ООП в разработке позволяет структурировать информацию по определённой иерархии, лучше понимать код и масштабировать его под различные задачи, внедрять изменения без необходимости переписывать весь код целиком и так далее.

Среды разработки

Интегрированная среда разработки (англ. integrated development environment - IDE) – это комплекс программных средств, предоставляющих набор функций и возможностей, для создания программного обеспечения. Среда разработки включает в себя текстовый редактор кода, транслятор (компилятор и/или интерпретатор), а также средства автоматизации сборки (относительно автономно от разработчика и его ПК).

## Краткая история сред разработки

Разработка с использованием ИСР стала возможно тогда, когда программы начали писать с помощью консолей и терминалов. Так, «Dartmouth BASIC», разработанный в 1964 году, стал первым созданным языком, имеющим свою собственную среду разработки, а также первым, созданным с помощью консоли. Данная ИСР была первой, интегрировавшей функции редактирования, управления файлами, компиляции и отладки способным, схожим с современными ИСР.

«Maestro I», разработанная компанией Softlab Munich в середине 70-х годов, стала первой в мире ИСР для программного обеспечения. Известно, что всего в мире было произведено около 22 тысяч установок Maestro I. Помимо этого, эта среда разработки была лидером в своей области во время 70-х и 80-х готов.

Ближе к 80-м годам появились графические пользовательские интерфейсы (англ. GUI). Благодаря их использованию в ИСР, процесс разработки стал ещё удобнее и эффективнее, т.к. это позволило использовать программистам визуальные инструменты для разработки интерфейсов и управления проектами. Примеры: «Microsoft Visual Studio» от Microsoft Corporation 1997 года выпуска, «Delphi» от Borland 1995 года.

Современные ИСР в большинстве случаев поддерживают несколько языков программирования, а также обладают множеством различных инструментов для более эффективного процесса разработки и управления циклом программного обеспечения.

Помимо этого, существуют онлайн-версии ИСР, известные как Web-IDE (Веб-ИСР) или Cloud IDE (Облачные ИСР). Хоть и не обладают таким же спектром функций, как их традиционные десктоп-аналоги, они поддерживают базовые функции ИСР (подсветка текста, поиск ошибок и т.д.), что позволяет разработчику продолжать работу на устройстве без установки ИСР.

Также существуют версии ИСР для мобильных устройств, таких как смартфоны или планшеты. Они имеют ограниченный набор инструментов, но компенсируют это своей мобильностью и доступностью.

По состоянию на 2023 год, самые популярные ИСР сегодня, согласно запросам в Google: Visual Studio, Visual Studio Code и Eclipse.

## Инструменты современных сред разработки

Интегрированные среды разработки, используемые сегодня, обладают широким диапазоном различных инструментов для разработчика.

Примеры:

* Подсветка синтаксиса – ИСР автоматически выделяют разными цветами языковые структуры, ключевые конструкции, а также синтаксические ошибки, что облегчают навигацию в коде программы
* Автоматическое написание кода – функция, предназначенная для ускорения процесса написания кода, добавляющая в код различные языковые конструкции в автоматическом режиме, что позволяет программисту не тратить время на их написание вручную. Некоторые ИСР поддерживают авто-написание с использованием ИИ и нейросетей
* Контроль версий – ряд современных ИСР поддерживает связь с системами контроля версий (такими как Git), что позволяет взаимодействовать с удалёнными репозиториями в облаке
* Отладка – ИСР могут производить отладку кода программы, с поддержкой установки точек остановы (англ - break points), отслеживанием шагов отладки и так далее
* Поиск и навигация по коду – ИСР позволяют находить определённые фрагменты кода в коде программы, а также файлы в файловой системе ПК
* А также многие другие

Вышеуказанные наборы инструментов присущи большинству современных используемых ИСР, что повышает их эффективность и универсальность, позволяя разработчику редактировать один и тот же код в различных ИСР.

Заключение

Таким образом, сегодня в мире существует огромное количество различных языков программирования и сред разработки. Все они обладают определённым набором характеристик, инструментов и других свойств, что обуславливает использование одних вместо других, делая одни более универсальными или наоборот, узкоспециализированными. Тем не менее, не существует универсальных примеров, подходящих для всех ситуаций – выбор языка программирования и/или среды разработки зависит от множества факторов: параметры конечного продукта, навыки разработчика, удобство и скорость использования, а также многие другие.

Следует осмысленно подходить к вопросу выбора языка/ИСР, чтобы конечная эффективность процесса разработки продукта была максимальной. Важно понимать, что в одних случаях лучше подходят одни примеры, а в других – другие.