**Программирование**, процесс составления упорядоченной последовательности действий (*программы*) для ЭВМ; научная дисциплина, изучающая программы для ЭВМ и способы их составления, проверки и улучшения.  
  
  Каждая ЭВМ является автоматом, состоящим из памяти, образуемой внешним и оперативным запоминающими устройствами, устройства управления (УУ) и арифметические устройства (АУ), в котором могут выполняться некоторые действия или операции. Память имеет вид занумерованной последовательности ячеек, в каждой из которых хранится порция двоичной информации в виде серии нулей и единиц. Автоматическая работа ЭВМ, управляемая программой, состоит из последовательности тактов. На каждом такте УУ выбирает из предписанной ему ячейки памяти порцию информации. Эта порция трактуется как команда, т. е. предписание АУ выполнить некоторую операцию. Обычно в ЭВМ выполнение операции состоит в том, чтобы из определённых ячеек памяти взять хранящуюся там информацию, передать её в АУ для выполнения над ней нужного действия, результат которого отправить в указанную ячейку памяти, и сообщить УУ номер ячейки следующей команды. Отдельные действия, совершаемые ЭВМ, весьма просты — это арифметические и логические операции, операции сравнения, переписывания порции информации и т.п. Т. о., составить программу для ЭВМ — это значит представить способ решения задачи в виде такой совокупности машинных команд (программы), чтобы они, будучи размещенными в памяти, поочерёдно выполняясь и вызывая одна другую, реализовали нужные вычисления.  
  
  Идея **Программирование** возникает ещё в школе при составлении «плана решения» арифметической задачи в виде серии вопросов. Существенное отличие реального **Программирование** от школьного опыта заключается в том, что программа, как правило, задаёт не одну, а несколько последовательностей действий (разветвлений), выбор между которыми зависит от значения промежуточных результатов решения задачи; выполняет некоторые группы команд многократно, автоматически определяя нужное число повторений; может предписанным образом сама себя менять или частично формировать в процессе своего выполнения.  
  
  Дополнительной особенностью **Программирование** является его трудность: размеры многих реальных программ исчисляются тысячами команд, а количество выполняемых ими действий — десятками миллионов. Такие объёмы в сочетании с элементарным характером машинных команд делают **Программирование** одновременно и очень сложным, и очень монотонным процессом.  
  
  Для преодоления этого противоречия **Программирование** придан характер многоэтапного процесса, каждый этап которого есть постепенная конкретизация и детализация плана решения задачи, полученного на предыдущем этапе. Кроме того, если правила описания плана решения задачи на некотором этапе будут точными, формальными и универсальными, т. е. применимыми к любой задаче, то тогда можно говорить о существовании некоторого языка **Программирование**, используемого при составлении программы.  
  
  *Языки программирования* как способ точного формулирования задачи на разных этапах подготовки её к решению на ЭВМ сыграли фундаментальную роль в развитии **Программирование** В частности, они позволяют трактовать **Программирование** как процесс перевода задания для ЭВМ, выраженного в некотором языке, на другой язык — «язык машины». Если найти и описать точные правила такого перевода, то эти правила, в свою очередь, можно запрограммировать на ЭВМ. Полученные программы, автоматизирующие процесс **Программирование**, называют трансляторами.  
  
  Процесс **Программирование** обычно состоит из следующих этапов:  
  
  
  
  Содержание каждого этапа **Программирование** можно пояснить на примере решения квадратного уравнения.  
  
  Исходная формулировка. Надо найти корни 50 квадратных уравнений вида *ax2* + *bx* + *c* =0, коэффициенты которых заданы в виде трёх таблиц *Ai*, *B*i и *Ci* (*i* = 1,..., 50).  
  
  Алгоритмическое описание задачи получается в результате полного математического исследования её постановки, выбора стандартных или поиска новых алгоритмов выполнения всех нужных вычислений, а также уточнения, какие исходные данные надо ввести в ЭВМ и какие результаты надо получить. В данном случае алгоритмическое описание может иметь такой вид: ввести в ЭВМ таблицы коэффициентов *Ai*, *B*i и *Ci*, решить каждое уравнение по общей формуле  
  
  
  
с исследованием дискриминанта *b2 —* 4*ac* для определения случая комплексных корней; для единообразия каждый корень уравнения выдавать как комплексное число *x* = *u* + *iv*, полагая в случае вещественных корней мнимую часть равной нулю.  
  
  Язык **Программирование** высокого уровня — главное средство составления программ для ЭВМ. Общая особенность этих языков — их независимость от системы команд конкретных ЭВМ и фразовая структура, что в сочетании с использованием т. н. служебных слов приближает их к естественным языкам. Фразы разделяются обычно точкой с запятой; соподчинённость фраз указывается с помощью «операторных скобок» начало и конец; фразы делятся на два типа — операторы и описания. Оператор является единицей действия в языке. Различают следующие их типы: операторы присваивания, производящие подсчёт по указанной формуле и присваивающие вычисленное значение заданной переменной величине; условные операторы, которые в зависимости от результата проверки заданного условия производят выбор одной или другой ветви вычислений; операторы цикла, обеспечивающие повторное выполнение группы операторов. В описаниях указываются свойства переменных величин и других обозначений, используемых в программе. Важным свойством является «процедурный» характер языка: для любой уже составленной программы, решающей некоторую частную задачу, можно ввести символическое функциональное обозначение. Текст этой программы вместе с её обозначением называется описанием процедуры или подпрограммой. Тогда при составлении новой программы всюду, где может потребоваться использование этой описанной процедуры, достаточно упомянуть её функциональное обозначение в виде оператора процедуры вместо переписывания полного текста подпрограммы.  
  
  В 70-х гг. 20 в. существует целое семейство таких языков **Программирование**: алгол-60, фортран для решения инженерных и научных задач, кобол для экономических расчётов, симула для **Программирование** математических моделей, более мощные языки алгол-68 и ПЛ/1, охватывающие все виды применений ЭВМ. Для всех из них существуют трансляторы, обеспечивающие автоматическое построение машинных программ для задач, выраженных в этих языках.



Машинно-ориентированный язык представляет программы в терминах команд ЭВМ, но выраженных в более удобной для употребления символике, нежели прямое двоичное представление. Он используется на промежуточном этапе процесса автоматической трансляции с языка более высокого уровня или же как язык **Программирование**, когда программа по существу сразу должна быть сконструирована в терминах машинных команд. В последнем случае роль языка высокого уровня часто играет язык блок-схем, когда структура программы, т. е. последовательность выполнения её «блоков», наличие разветвления и повторяющихся участков показываются в графической форме, а функции каждого блока записываются в произвольной текстовой форме. Ниже следует пример блок-схемы решения квадратного уравнения:  
  
  
  
  После составления программы неотъемлемым этапом **Программирование** является «отладка» программы, т. е. обнаружение и исправление ошибок, допускаемых при **Программирование** Основное средство отладки — т. н. отладочные запуски, когда в программу добавляются дополнительные «измерительные» действия, позволяющие по ходу выполнения программы на ЭВМ выдавать «протокол» её работы (порядок выполнения команд, значения промежуточных результатов и т.п.). Исследование протокола позволяет судить о том, в какой степени программа соответствует замыслу программиста.



Широкое внедрение вычислительной техники в различные сферы деятельности человека стимулировало развитие автоматизированных методов и инструментальных средств создания прикладного программного обеспечения (ПО). Производство современного ПО происходит на фоне высоких требований к его качеству, так как сложность и ответственность выполняемых им функций непрерывно возрастает. Создание качественной программы, как и любого другого продукта, зависит не только от опыта разработчика, но и от инструмента, которым он пользуется. “**Идеальной” технологией программирования была бы такая технология, которая по некоторому неформальному описанию объекта программирования автоматически генерировала бы текст синтаксически и семантически корректной программы.**

**Разработка любого программного средства может быть представлена как процесс, состоящий из ряда последовательных преобразований одного описания решаемой задачи в другое**, начиная от постановки задачи и заканчивая программой, реализованной в кодах конкретной ЭВМ.

**Все время существования программного средства, от зарождения идеи по его созданию до завершения его эксплуатации, обычно определяют как жизненный цикл.** Укрупнено можно выделить пять наиболее важных этапов жизненного цикла программного средства (ЖЦ ПС): **спецификацию (10%),** **проектирование (10%),** **кодирование (10%),** **отладку (20%) и сопровождение (50%).** В скобках записаны относительные затраты ресурсов на создание ПС.

По затратам времени, человеческих и машинных ресурсов все эти этапы не одинаковы. Наиболее “дорогими”, в этом смысле, являются этапы, связанные **с поиском ошибок в программах**. Затраты ресурсов на них могут быть равными, или даже превосходить совокупные затраты ресурсов на остальных этапах. **В стандарте DOD-STD-2167-A около 30% требований, документов и соответствующих им процессов непосредственно связаны с отладкой, тестированием и испытаниями программ**. Данный стандарт является обязательным при выполнении заказов Министерства обороны США.

Эти затраты быстро увеличиваются при возрастании требований к качеству ПС. По оценкам, приведенным в работе [111], **на долю устранения ошибок и сопровождение ПС приходится почти 75% всех затрат**. Следует учитывать, что значительная часть работ, выполняемых на этапе сопровождения, связана с поиском и устранением оставшихся в программе ошибок.

Ретроспектива развития методов и средств автоматизации программирования в этом отношении говорит сама за себя. Первоначально все усилия были сосредоточены на этапе программирования, что позволило решить ряд проблем, связанных с ограничениями функциональных возможностей существующих к тому моменту языков программирования.

На этом этапе следует выделить развитие специализированных языков программирования, таких как КОБОЛ, СИМУЛА, СИМСКРИПТ [40], АСПИД [140], АНАЛИТИК [157] и т.д.

**Специализированные или проблемно-ориентированные языки** [34], построенные для решения конкретных задач, несомненно упрощают процесс разработки программных продуктов в выбранной предметной области, но в то же время для новых языков характерна проблема преемственности, а для специализированных - ограниченности области применения.

Постепенное усложнение решаемых задач, а вместе с ними кодов и объемов разрабатываемых программных средств, заставило критически переосмыслить сложившийся к концу 60-х годов стиль и технику программирования. В связи с чем, начиная с 70-х годов усилиями рабочей группы Технического комитета по программированию Международной федерации по обработке информации (ИФИП), состоящей из таких известных ученых как Н. Вирт, Д. Грис, Э. Дейкстра, У. Дал, Д. Парнас, Ч. Хоар (руководимой профессором В. Турским), формируются основы методологии и теории программирования.

В среде программистов утверждается***принцип структурного программирования*,** как наиболее производительного и привлекательного стиля программирования [26, 41]. Развитие концепции *структуризации* привело к осознанию необходимости структуризации данных [2], что и определило появление в языках программирования *механизмов абстрагирования типов данных* (Клу [166], Модула [25] и д.р.). Последнее легло в основу ***техники модульного программирования***, содержательной основой которой является интерпретация типа как алгебры над множеством объектов и множеством операций, а модуля как программного эквивалента абстрактного типа.

В модульном программировании акцент делается на разбиение программы на модули таким образом, чтобы данные (обрабатываемые модулем) были спрятаны в нем. Эта доктрина, известная как “принцип ограничения доступа к данным”, в значительной степени повысила модифицируемость и эффективность порождаемого кода.

Эволюция техники модульного программирования привела к появлению **объектно-ориентированного стиля программирования** [154, 158, 170], который во многом унифицировал процесс создания ПС. К достоинствам этого метода относится то, что в нем более полно реализуется технология структурного программирования, облегчается процесс создания сложных иерархических систем, появляется удобная возможность создания пользовательских библиотек объектов в различных областях применения.

Параллельно с развитием процедурного стиля программирования в начале 70-х годов появляются непроцедурные **языки логического программирования и программирования искусственного интеллекта** (LISP [51, 122], PROLOG [15, 114], РЕФАЛ [149], ПРИЗ [61,144]) . Программа в таких языках представляет собой совокупность *правил* (определяющих отношения между объектами) и *цели* (запроса). Процесс выполнения программы трактуется как процесс установления общезначимости логической формулы по правилам, установленным семантикой того или иного языка. Результат вычислений является побочным продуктом процедуры вывода. Такой метод являет собой полную противоположность программирования на каком-либо из процедурных языков. Сегодня PROLOG - язык, предназначенный для программирования приложений, использующих средства и методы искусственного интеллекта, создания экспертных систем и представления знаний.

В 80-х годах исследование причин неудач при реализации больших программных проектов [167] показало, что число ошибок в спецификациях на программы значительно превышает их количество в программных кодах. **Так около 56% ошибок допускаются на этапе формулировки требований к программе при этом расходуется в среднем 82% всех усилий, затраченных коллективом на устранение ошибок проекта. В то время как на этапе кодирования программ допускается соответственно 7% ошибок и тратится 1% усилий на их ликвидацию [167].** В это время формулируется тезис о том, что целью программирования является не порождение программы как таковой, а создание технологических условий, когда разрабатываемое программное обеспечение легко адаптируется к новым обстоятельствам и новому пониманию решаемой задачи [20, 100]. Р. Хемминг так формулирует этот тезис: “**Здравая вычислительная практика требует постоянного исследования изучаемой задачи не только перед организацией вычислений, но также в процессе его развития и особенно на той стадии, когда полученные числа переводятся обратно и истолковываются на языке первоначальной задачи” [148].**

Перечисленные выше причины привели в середине 80-х годов к осознанию необходимости реализации интегрированного окружения поддержки всего жизненного цикла ПС и, в первую очередь, этапа проектирования ПС, что обусловило появление **инструментальных средств автоматизации проектирования программных систем (CASE-технологий**) [155, 171, 172].

Первоначально CASE-средства были ориентированы на решение задач автоматизированного сбора информации по предметной области и проектированию будущего ПС, что позволяет экономить время при создании ПС за счет более тщательного анализа исходных требований и лучшего начального планирования программы. Впоследствии в CASE-средствах 2-го поколения полностью или частично были автоматизированы такие важные составляющие жизненного цикла ПС как моделирование информации предметной области; программирование; тестирование, отладка ПС и измерение качества; поддержка документирования; сопровождение.

Применение CASE-инструментов позволяет в значительной степени снизить трудоемкость создания ПС, а в отдельных случаях заменить программирование автоматическим синтезом программ.

**Таким образом, развитие методов автоматизации разработки ПС происходит на различных основах (модульное программирование, объектно-ориентированный подход, логическое программирование, CASE-технологии), которые, так или иначе, развивают концепции структуризации в программировании.** *Структуризация способствует проведению эффективной декомпозиции проекта, что позволяет получать как целостное представление о ПС, так и его деталях.* Однако, несмотря на многочисленные разработки в этой области, в целом проблема автоматизации разработки ПС остается нерешенной по многим причинам как методологического, так и практического характера.

В последнее время в связи с совершенствованием технических средств отображения информации утверждается **новый графический подход** к решению проблемы автоматизации разработки ПС, основанный на идее привлечения визуальных форм представления программ, в большей степени соответствующих образному способу мышления человека. Применение графических методов обещает кардинально повысить производительность труда программиста. Кроме того, графическая форма записи по сравнению с текстовым представлением программ обеспечивает более высокий уровень их структуризации, соблюдение технологической культуры программирования, предлагает более надежный стиль программирования [23].

В настоящее время известно достаточно большое количество удачных **инструментальных средств визуализации программирования**. Прежде всего это относится к визуальным средствам разработки экранных форм, меню и других элементов программы (Microsoft Windows 3.0, Visual BASIC [38, 59,133] и т.д.), средствам автоматизации проектирования программного обеспечения (CASE-средствам), средствам быстрой разработки приложений для информационных систем (Visual FoxPro), текстовым и графическим редакторам, издательским системам и т.д.

В соответствии с “формулой” Н. Вирта [26] программу можно интерпретировать как “Алгоритм + Структуры данных”. **Визуальные формы представления алгоритмов программ** известны достаточно давно это  
Р-технология программирования И. В. Вельбицкого [21], сети Петри [19], наконец, обычные блок-схемы можно считать в определенном смысле графической формой задания алгоритма. Общим для перечисленных выше способов представления алгоритмов является использование “языка” теории графов.

Одним из основных факторов повышения эффективности и надежности программирования можно считать придание образности формам спецификации данных и описания алгоритма. В этом смысле главный недостаток существующих технологий программирования заключается в преимущественно текстовых формах представления основных компонент программы, что делает программу невыразительной и чрезвычайно затрудняет ее восприятие человеком.