

Парадигмы программирования



Парадигма программирования - исходная концептуальная схема постановки задач и их решения; вместе с языком, ее формализующим.

Парадигма формирует стиль программирования.

Парадигма (παράδειγμα, «пример, модель, образец») — совокупность фундаментальных научных установок, представлений и терминов, принимаемая и разделяемая научным сообществом и объединяющая большинство его членов. Обеспечивает преемственность развития науки и научного творчества.

Парадигмы программирования

Парадигма является инструментом грамматического описания фактов, событий, явлений и процессов, возможно, не существующих одновременно, но интуитивно объединяемых в общее понятие.

Парадигма представляет (и определяет) то, как программист видит выполнение **плана** решения поставленной задачи (программы).

Например, в объектно-ориентированном программировании программист рассматривает программу как набор взаимодействующих объектов, тогда как в функциональном программировании программа представляется в виде цепочки вычисления функций.

Парадигмы программирования

Важно отметить, что парадигма программирования не определяется однозначно языком программирования; практически все современные языки программирования в той или иной мере допускают использование различных парадигм (мультипарадигмальное программирование).

Так на языке Си, который не является объектно-ориентированным, можно работать в соответствии с принципами объектно-ориентированного программирования, хотя это и сопряжено с определёнными сложностями; функциональное программирование можно применять при работе на любом императивном языке, в котором имеются функции.

Парадигма программирования

Первым языком программирования высокого уровня считается язык Plankalkül (*исчисление планов*) разработанный немецким инженером Конрадом Цузе ещё в 1942 году.

Однако транслятора для него не существовало до 2000 г.

Математики описывали задачу на этом языке, а потом специально обученные техники переводили ее в машинный код.

Математик → Программист

С появлением первых программируемых ЭВМ (EDSAC, МЭСМ, EDVAC) в 1948 - 1951 гг. начались работы по разработке планов управления ЭВМ (методов программирования):

1950 – Операторный метод (А.А. Ляпунов)

1951 – Алгебраический метод (Грейс Хоппер)



Математик → Программист

Ресурс, существенный почти для всех реальных исполнителей - это время.

Важное свойство ЭВМ как исполнителей - способность выполнять элементарные действия исключительно быстро (порядка долей микросекунды на действие).

Не менее важное свойство ЭВМ - способность хранить огромные объемы данных (в оперативной памяти - мегабайты; на внешней - практически неограниченно).

Математик → Программист

Для ЭВМ как исполнителей характерны два вида ресурсов - память и процессор.

Память реализует первую из двух названных фундаментальных способностей - служит для хранения данных.

Процессор реализует вторую из названных способностей - служит для выполнения действий, предусмотренных в планах.

Процессор характеризуется определенным набором допустимых действий (операций, команд). Действия из этого набора считаются элементарными (в плане не нужно заботиться о способе выполнения таких действий).

Математик → Программист

Две названные способности связаны - выполнение достаточно сложного плана требует его хранения в доступной исполнителю памяти. В свою очередь, реализация хранения требует способности выполнять план (данные нужно размещать, перемещать, делать доступными).

План для такого исполнителя, как ЭВМ, должен в итоге сводиться к указанию конечной последовательности элементарных действий.

Такой план в настоящее время называют программой.

Математик → Программист

Людей как исполнителей характеризует прежде всего наличие у них модели реального мира, в достаточной степени согласованной с моделью мира у создателя плана.

Поэтому в плане для людей можно указывать цели, а не элементарные действия.

Морис Уилкис

1950 год - *Морис Уилкис* (конструктор *EDSAC*) ввел систему мнемонических обозначений для машинных команд, названную языком **ассемблера** (англ. *assembler* — сборщик).



В июне 1951 Морис Уилкс представил доклад "наилучший метод конструирования автоматической машины", который стал пионерской работой по основам **микропрограммирования**. Свою идею микропрограммирования Морис Уилкс реализовал в 1957 году при создании машины EDSAC-II.

М.Уилкс совместно с Д.Уиллером и С.Гиллом в 1951 году написали первый учебник по программированию "Составление программ для электронных счетных машин" (русский перевод - 1953 год).

Алексей Андреевич Ляпунов

В 1950 году А.А. Ляпуновым был предложен **операторный метод** для описания программ.

В течении 1950—1953 годах впервые был создан способ представления программ на обозримом уровне.

Вместо неэффективного для человека задания программ в машинных кодах А.А. Ляпунов предложил формализованное представление высокого уровня. Особенно важным было то, что **операторный метод** позволял создать теорию синтаксических структур программ.



Леонид Витальевич Канторович

В 1953—1954 годах Л.В. Канторович разработал технологию крупноблочного программирования, которая также давала обозримое описание программ и обеспечивала степень формализации, достаточную для исследования синтаксических структур программ и создания программирующих программ.



Это направление отходило от традиционного (фон-Неймановского) программирования, что, естественно, затрудняло развитие этих идей. Школа следовала математическому направлению, которое связывается сейчас, в частности, с функциональным стилем и проявляется также в других подходах, когда проблемы программирования все теснее начинают увязываться с математическими представлениями, результатами.

Леонид Витальевич Канторович

Характерной особенностью крупноблочных систем являлось то, что они оперировали не с индивидуальными числами и символами, а с величинами — укрупненными агрегированными информационными объектами.

Такие укрупненные структуры данных (матрицы, векторы, последовательности, деревья, схемы и т. д.) выступали как целое в вычислительных планах; стандартные способы обработки отдельных элементов величин выполнялись автоматически на нижних уровнях. Это вносило иерархическую структуру в языки программирования, освобождая верхние уровни от ненужной детализации.

Леонид Витальевич Канторович

Отметим следующий принципиальный момент. С каждой величиной при вводе ее в машину связывались три характеристики, выделялись три ее стороны:

1. имя,
2. справка (информация о типе и структуре величины, о ее размещении в машинной памяти),
3. запись (т. е. значение, фактическое представление элементов величины, ее денотат).

Так очень естественно уже на первых шагах были отчетливо введены в рассмотрение синтаксический, смысловой и интерпретационный уровни информационных объектов, и объект рассматривался “объемно”, как целостное единство этих компонент.

Мама программирования

Хоппер и ее группа принялись за разработку компилятора, который позволял бы людям, занятым в бизнесе, программировать на языке, по возможности приближающемся к обычному английскому языку.

Они язык (словарь основных терминов), на котором можно было бы писать команды типа COMPARE (сравнить) или TRANSFER (переслать). Через несколько месяцев работы Хоппер и ее коллеги выделили список примерно из 30 слов, которые стали «рабочими лошадками» обработки данных.

Впоследствии вошедшие во все ЯВУ.



ЯВУ



1954-57 г., Дж.Бэкус

- FORTRAN
- язык ассемблера - ПП (Ляпунов)
- машинные коды
- программирование переключателей

```
0000 0A 12 1F 4B C3 E0 EE F1
0008 C3 1D 23 17 F2 00 0C 0D
0010 ...
```

```
MOV     AX, [ARG1]
ADD     AX, [ARG2]
MOV     [RES], AX
JMP     NEXT
ARG1:   DB      10
ARG2:   DB      20
RES:    DB      0
NEXT:   ...
```

```
S = 0
DO 10 I=1,10
S = S + I*I
10  CONTINUE
```

- Первый язык программирования высокого уровня – ФОРТРАН – был создан Дж.Бэкусом, чтобы математики могли программировать на уровне формул.

Императивный стиль

- Традиционно под планом работы ЭВМ (программой) понимают последовательность операторов (команд, выполняемых компьютером). Такой стиль программирования принято называть **императивным**.
- Програмируя в **императивном стиле**, программист должен объяснить компьютеру, что нужно делать, чтоб решить задачу.
- Императивная программа очень похожа на приказы, выражаемые повелительным наклонением в естественных языках, то есть это последовательность команд / логических переходов, которые должен выполнить компьютер.
- Основные конструкции:
 - Манипулирование ячейками памяти
 - Оператор присваивания
 - Явные операторы передачи управления
 - Циклы, условный оператор

Императивные языки

- Основаны на фон неймановской модели вычислений.
- Решая задачу, императивный программист вначале создает модель в некоторой формальной системе, а затем переписывает решение на императивный язык.
- Для человека рассуждать в терминах компьютера довольно неестественно. Во-вторых, последний этап этой деятельности (переписывание решения на язык программирования), что по сути не имеет отношения к решению исходной задачи.
- Часто императивные программисты даже разделяют работу в соответствии с двумя описанными выше этапами. Одни люди, постановщики задач, придумывают решение задачи, а другие, кодировщики, переводят это решение на машинный язык.

Декларативный стиль

- программа представляет собой совокупность **утверждений**, описывающих фрагмент предметной области или сложившуюся ситуацию; описывается результат (его свойства), а не методы его достижения.
- программируя в декларативном стиле, программист должен описать, что нужно решать.
- В основе **декларативных языков** лежит формализованная человеческая логика. Человек лишь описывает решаемую задачу, а поиском решения занимается императивная система программирования. В итоге получаем значительно большую скорость разработки приложений, значительно меньший размер исходного кода, легкость записи знаний на декларативных языках, более понятные, по сравнению с императивными языками, программы.

Уровни языков программирования

- Известна классификация языков программирования по их близости либо к машинному языку, либо к естественному человеческому языку.
- Те, что ближе к компьютеру, относят к языкам **низкого уровня**, а те, что ближе к человеку, называют языками **высокого уровня**.
- В этом смысле декларативные языки иногда считают языками **сверхвысокого** или **наивысшего** уровня, поскольку они очень близки к человеческому языку и человеческому мышлению.

Парадигма программирования

- **Парадигма программирования** - исходная концептуальная схема постановки проблем и их решения; вместе с языком, ее формализующим, парадигма формирует **стиль** программирования.
- Парадигма является инструментом грамматического описания фактов, событий, явлений и процессов, возможно, не существующих одновременно, но интуитивно объединяемых в общее понятие. Т.е., парадигма представляет (и определяет) то, как программист видит выполнение программы.
- *Например*, в объектно-ориентированном программировании программист рассматривает программу как набор взаимодействующих объектов, тогда как в функциональном программировании программа представляется в виде цепочки вычисления функций.

Основные виды парадигм

- Процедурная
 - Структурная
 - Параллельная
 -
- Функциональная
- Логическая
 - Объектно-ориентированная
 - Аспектно-ориентированная

Семантика языков

Императивные языки:

Оперируют состоянием памяти.
Действие операторов – изменяет состояние.

Функциональные языки:

Оперируют с данными (подход ориентированный на данные). Применение функций изменяет данные

Логические языки:

Оперируют с пространством поиска решений.
Программа задает множество возможных переходов в пространстве поиска.

Процедурное программирование

- Программа состоит из структур данных (объектов обработки) и алгоритма (метода обработки).
- Программист должен в явном виде описать все вычисления, которые должен проделать компьютер.
- Для управления процессом выполнения используются следующие конструкции, последовательность, ветвление, цикл и вызов подпрограммы.
- Эта парадигма является самой старой. Она развивалась по мере появления новых концепций в языках программирования: трансляция (Ассемблер, Fortran, Cobol), типизация (Pascal), модули (Modula), специализация на конкретной области применения (RPG, Clipper) и универсальность (PL/I, C, Ada).

Функциональное программирование

- Сформировалось как дань математической направленности при исследовании и развитии искусственного интеллекта и освоении новых горизонтов в информатике.
- Единственной управляющей конструкцией является вызов функции. В языке, реализующем функциональную парадигму, существует некоторое множество базовых функций, и все другие функции строятся из базовых функций с помощью композиции.
- Теоретической основой является лямбда-исчисление и теория рекурсивных функций.
- В настоящее время существуют сотни функциональных языков программирования, ориентированных на разные классы задач и виды технических средств: Lisp, Haskell, ...

Логическое программирование

- Возникло как упрощение функционального программирования для математиков и лингвистов, решающих задачи символьной обработки.
- Вместо описания алгоритма решения задачи описывается мир задачи, какие имеются объекты, их свойства и отношения между ними.
- За основу описания берутся отношения между объектами.
- Логическая программа представляет собой набор отношений, которые называются фактами, и правил, на основании которых могут быть получены новые отношения. Она не задает никакого процесса вычислений. Это своего рода база данных (БД) о предметной области задачи. Ее применение инициализируется запросом. Поиск ответа на запрос заключается в попытке логического вывода запроса на основании фактов и правил, имеющихся в БД. Поиск решения выполняется специальной программой - интерпретатором.
- Основной (самый популярный) язык – Prolog, с множеством диалектов.
- Другие (менее популярные) языки: Datalog, Mercury, Oz.

Объектно-ориентированный подход к процедурному программированию

- Мир задачи описывается как совокупность объектов, обладающих некоторыми свойствами и вступающих во взаимодействие. За основу описания берутся не отношения, а сами объекты.
- Объект состоит из данных (внутренняя структура объекта) и методов (набор операций, применимых к данному объекту). С помощью понятия класс, объединяющего совокупности объектов с похожими свойствами и поведением, выстраиваются иерархия объектов.
- Каждый объект наследует свойства своего класса и может иметь свои собственные свойства. Программа в ООП - это совокупность объектов, обменивающихся между собой сообщениями.
- Языками ООП являются Smalltalk, C++, ...

Развитие парадигм

- По мере возрастания сложности задач, произошло расслоение средств и методов программирования. Выделились основные стили (внутренне концептуально согласованные совокупности средств, базирующихся на некоторой логике построения программ).
- Различным классам задач и методов соответствуют различные методы/логики построения программ, часто несовместимые между собой.
- Современные специализированные стили программирования:
 - Машинно-ориентированное
 - Системное
 - Высокопроизводительное
 - Трансформационное
 - Прикладное
 - Теоретическое
 - Концептуальное

Жизненный цикл программного обеспечения

- Обоснование требований
- Спецификации
- Кодирование
- Тестирование
- Внедрение
- Сопровождение

Жизненный цикл парадигм

- Эволюция парадигм программирования происходила в связи с потребностью в *разделении ответственности* (separation of concerns): артефактов, концепций или особенностей, интересующих участников проекта разработки ПО.
- Зона ответственности может быть функциональной (например, добавление элемента к списку) или нефункциональной (например, быстродействие, безопасность и т.д.), либо относиться к этапам жизненного цикла разработки ПО, например к проектированию.

Жизненный цикл

- Разделение ответственности остается главным принципом программной инженерии, а его основа — общеизвестный метод снижения сложности с помощью абстрагирования.
- Процедурные языки изолируют и абстрагируют зоны ответственности в виде процедур и функций, тогда как в объектно-ориентированных языках, которые выросли из процедурных, в качестве абстракций используются классы и объекты.

Жизненный цикл

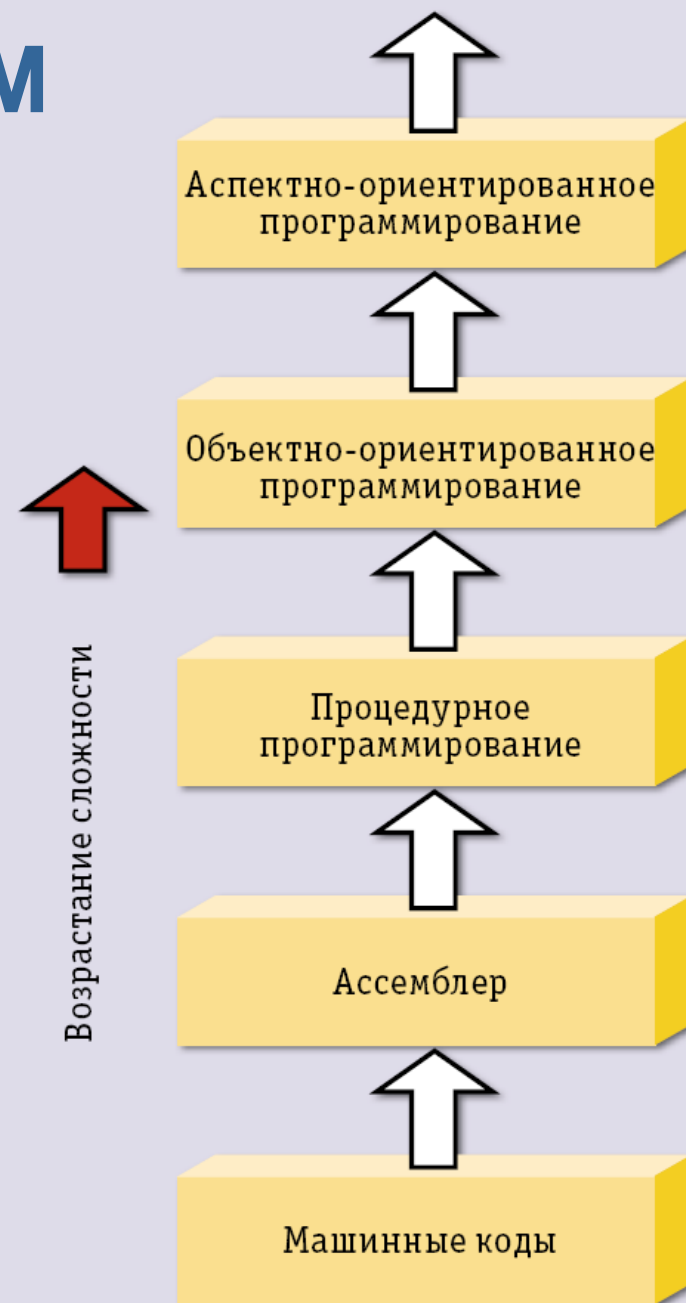
- Типичная программа представляет собой набор взаимосвязанных друг с другом компонентов (например, отвечающих за управление транзакциями), обычно имеющих взаимозависимости по тем или иным функциям.
- Такое взаимосоединение называется сквозной функциональностью (cross-cutting concern), и аспектно-ориентированное программирование занимается ее систематизацией, преобразованием в отдельные модули и их управлением.

Эволюция парадигм

- Как узнать, когда появляется потребность в новой парадигме? Какими механизмами абстракции она должна обладать? Как оценивать новые парадигмы?
- Рассмотрим базовые принципы эволюции парадигм программирования в **процедурном** подходе.

Эволюция парадигм

- Принцип эквивалентности.
- Принцип превосходства.
- Принцип связанных абстракций.
- Принцип ограниченной видимости.



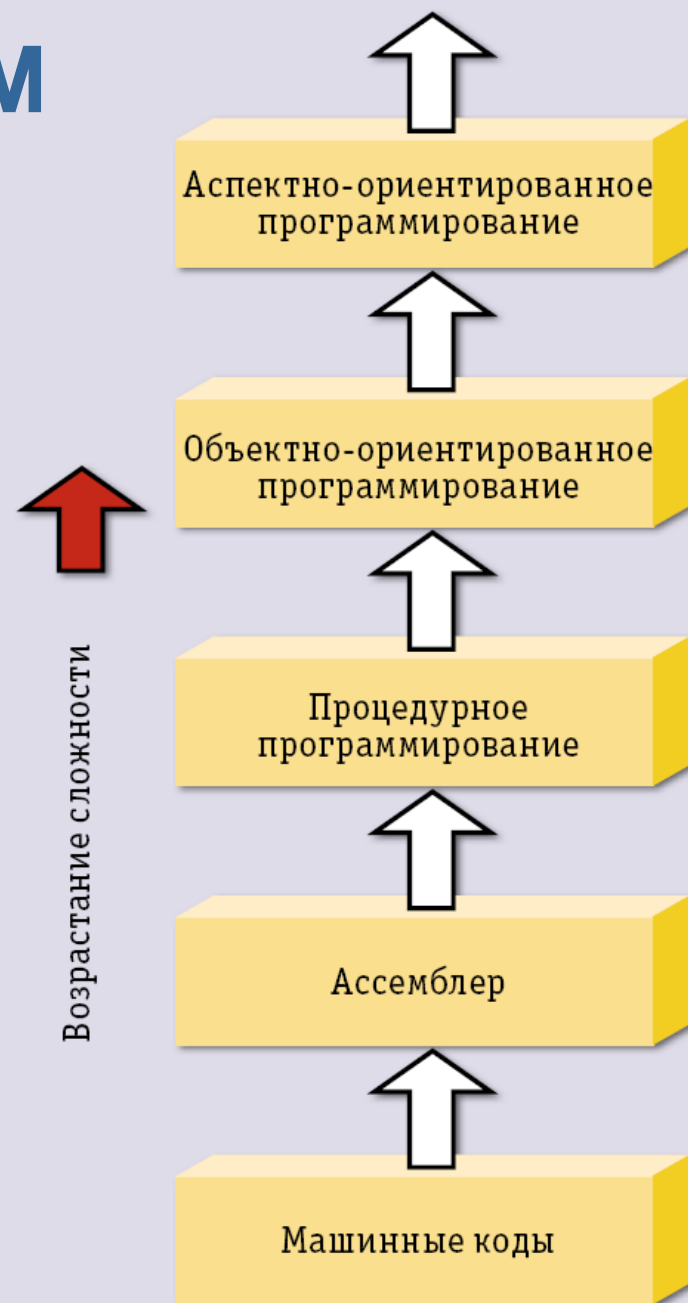
Эволюция парадигм

Принцип ограниченной видимости

- структурное программирование, лишенное переходов, было изобретено, чтобы удовлетворить потребность в интеллектуальном и семантическом контроле над управляющими структурами (ассемблер не давал возможности ощутить преимущества программирования без Goto, инкапсуляции и процедурных абстракций);
- ограничения процедурного программирования были связаны с невозможностью признать классы, наследование и полиморфизм в качестве более удачных механизмов абстрагирования;
- объектно-ориентированному программированию были свойственны ограничения в связи с неспособностью осознать преимущества инкапсуляции сквозной функциональности.

Эволюция парадигм

- Принцип эквивалентности.
- Принцип превосходства.
- Принцип связанных абстракций.
- Принцип ограниченной видимости.
- Принцип порога полезности



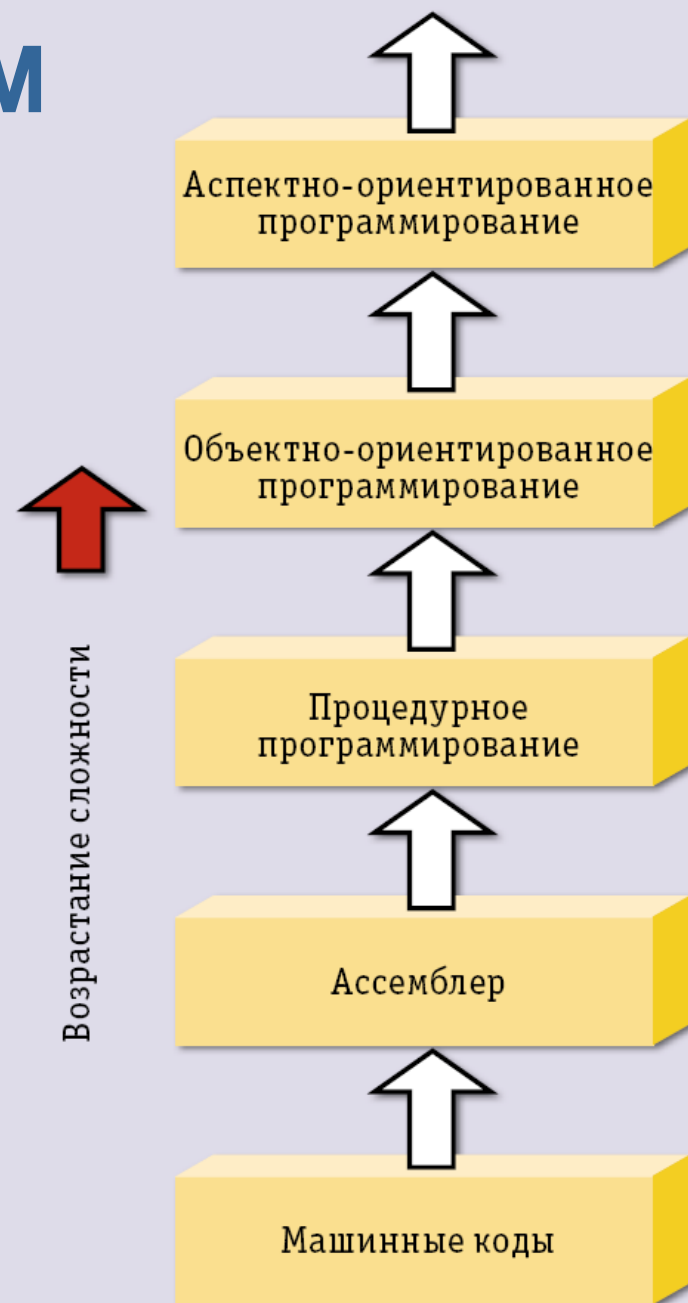
Эволюция парадигм

Принцип порога полезности

- порог полезности для программирования на ассемблере был достигнут, когда на первый план вышли потребности переносимости, понятности и сопровождаемости кода;
- порог полезности процедурного программирования был достигнут, когда стала первоочередной потребностью в уменьшении семантического пробела между задачами реального мира и программными моделями наряду с потребностью в инкапсуляции данных, а также в более развитых абстракциях для управления данными и алгоритмами;
- порог полезности ООП был достигнут, когда стала жизненно важной потребностью в инкапсуляции и управлении запутанным и разбросанным кодом.

Эволюция парадигм

- Принцип эквивалентности.
- Принцип превосходства.
- Принцип связанных абстракций.
- Принцип ограниченной видимости.
- Принцип порога полезности
- Принцип продления срока жизни
- Принцип неизбежности сложностей



Эволюция парадигм

Принцип неизбежности сложностей

Каждая новая парадигма программирования приносит с собой неизбежные сложности, связанные с переходом на новые механизмы абстракции и программирования.

Этот принцип хорошо иллюстрируется проблемами, обнаруженными в предыдущих парадигмах, — например, с ООП связаны аномалии наследования, такие как «проблема йо-йо» (чрезмерная размытость сильно связанного кода — чем глубже наследование, тем сложнее отследить поведение конечных классов) и трудности тестирования в контексте динамического связывания.

Процедурное программирование создало проблемы, вызванные возможностью доступа к одной и той же области памяти через разные имена переменных, а во многих процедурных языках отсутствовала простая возможность выполнения низкоуровневых вычислений, стандартных для ассемблера.

Парадигмы программирования

21 век - объектно-ориентированное программирование (ООП).

За какие-то 15 лет оно воплотилось в господствующую религию, подчинившую умы миллионов программистов.

ООП - стандарт де-факто.

С использованием объектно-ориентированной методологии (ООМ) осуществляется разработка огромного числа программных проектов, а ООП является основным инструментом для построения кода.



Парадигмы программирования

Повышение эффективности кодирования осуществляется путем автоматизации процессов объектно-ориентированного проектирования за счет применения соответствующих CASE средств. Наиболее известным инструментом проектирования является Rational Rose.

Языки программирования, методологии, приложения, операционные системы, базы данных... - кругом "объективная реальность".

ООП программирование

Объектно-ориентированное программирование (ООП) — парадигма программирования, основанная на представлении предметной области (и/или проблемной области) в виде информационных моделей на основе системы взаимосвязанных абстрактных объектов и их реализаций.

Основной проблемой процедурного программирования считается то, что данные и функции их обработки не были связаны.

Это вносило некоторую сложность в разработку программы. С появлением концепции ООП появилась новая структура «произвольных» данных — Класс.

ООП программирование

Это по сути дела тип данных, внешне похожий на структуру (в языке Си) или запись (в Pascal-е), в котором кроме данных (свойства) также содержались функции их обработки (методы).

При этом появляются такие свойства как наследование, полиморфизм и инкапсуляция.

ООП программирование

Объектно-ориентированная парадигма программирования не нова. Её истоки восходят к Симуле-67, хотя первая полная реализация была в Smalltalk-80.

ООП стало популярным во второй половине 80-х в таких различных языках, как Си-плюс-плюс, Objective C (другое расширение C), Object и Турбо Паскаль, CLOS (ОО-расширение Лиспа), Eiffel, Ада (в её последних воплощениях) и недавно - в Яве.

ООП программирование

Ключевые черты ООП хорошо известны.

Первая — это определение классов — пользовательских типов данных, инкапсулирующих своё содержимое и реализующих некоторые операции или методы над ним. Классы обычно являются основой модульности, инкапсуляции и абстракции данных в языках ООП.

ООП программирование

Вторая ключевая черта, — **наследование** — есть способ определения нового типа, наследуя элементы (содержание и методы) существующего и модифицируя или расширяя их. Это способствует выражению специализации и генерализации.

Третья черта, известная как **полиморфизм**, позволяет ссылаться на объекты различных классов (обычно внутри некоторой иерархии) однородным образом. Это делает классы ещё более удобными и делает программы, основанные на них, легче для расширения и поддержки.

ООП программирование

Классы, наследование и полиморфизм — фундаментальные свойства, требуемые от языка, претендующего называться объектно-ориентированным. (Языки, не имеющие наследования и полиморфизма, но имеющие только классы, обычно называются основанными на классах).

Различные ОО языки используют совершенно разные подходы. Мы можем различать ОО языки, сравнивая механизм контроля типов, способность поддерживать различные программные модели и какие объектные модели они поддерживают.

ООП программирование

Важнейшие принципы

Абстракция данных Объекты представляют собою неполную информацию о реальных сущностях предметной области. Их модели адекватны решаемой задаче, работать с ними намного удобнее, чем с низкоуровневым описанием всех возможных свойств и реакций объекта.

Наследование Наследованием называется возможность породить один **класс** от другого с сохранением всех свойств и методов класса-предка (иногда его называют суперклассом) и добавляя, при необходимости, новые свойства и методы. Наследование призвано отобразить такое свойство реального мира, как иерархичность.

ООП программирование

Важнейшие принципы

Полиморфизм

Полиморфизмом называют явление, при котором классы-потомки могут изменять реализацию метода класса-предка, сохраняя его **сигнатуру** (таким образом, сохраняя неизменным интерфейс класса-предка). Это позволяет обрабатывать объекты классов-потомков как однотипные объекты, не смотря на то, что реализация методов у них может различаться.

ООП программирование

Важнейшие принципы

Инкапсуляция — это принцип, согласно которому любой класс должен рассматриваться как чёрный ящик — пользователь класса должен видеть и использовать только интерфейс (от английского interface — внешнее лицо, т. е. список декларируемых свойств и методов) класса и не вникать в его внутреннюю реализацию. Этот принцип (теоретически) позволяет минимизировать число связей между классами и, соответственно, упростить независимую реализацию и модификацию классов.

ООП программирование

Многие современные языки специально созданы для облегчения объектно-ориентированного программирования.

Следует отметить, что можно применять техники ООП и для не объектно-ориентированного языка и наоборот.

Объектно-ориентированные языки программирования

C++ (расширение C)

Delphi (расширение Pascal)

MS Visual Basic (расширение Basic)

Ada, Java, C#, Perl, Python, PHP и др.

Следует также отметить язык программирования Оберон-2, разработанный под руководством Н.Вирта. Данный язык обладает простой структурой, компактен. Он сохраняет понятие типа, используемое в процедурном подходе. Для придания языку свойств объектного, предложен механизм связанных процедур.

Другие языки

- Фортран – далее APL (его название иногда расшифровывают как A Programming Language, но чаще Array Processing Language) (APL/360).
- Функциональное программирование: SECD машины Ландин's (1964)

Другие языки

- IAL (International Algebraic Language) - Алгол 58.
- Особый интерес представляет процедура разработки языка Алгол, которая привлекла к себе выдающихся ученых. Поколению программистов, воспитанных на языках Си или Java, могут показаться странными те надежды, которые возлагались на языки лет 40 назад.
- В них видели не столько прикладной инструмент программирования, сколько — не без излишней доверчивости по отношению к возможностям искусственного интеллекта — способ общения с машиной.
- Возможно, поэтому, а может быть, по причине большей, чем сегодня, гуманитарной составляющей в сознании программистской общности в центре внимания последней оказались работы лингвиста Ноама Хомского, изучавшего синтаксические особенности естественных языков. Хомский отличался значительным своеобразием и несомненной харизмой.

Другие языки



- Сегодня трудно сказать с полной уверенностью, что работы Хомского действительно серьезно повлияли на развитие языков программирования, но одно явное следствие, безусловно, есть — это нотация Бэкуса-Наура. Джон Бэкус и Питер Наур разработали формальный метод описания языков программирования, приобретший популярность благодаря Алгол: любой учебник по этому языку включал описание на BNF.

Другие языки

■ Сегодня Алгол интересен, пожалуй, только с исторической точки зрения, но, как ни странно, его непосредственный родственник JOVIAL живет и здравствует. Язык JOVIAL был создан Джулиусом Шварцем на IAL. Отсюда и аббревиатура JOVIAL (Jules Own Version of the International Algorithmic Language).

■ Долголетие это языка объясняется тем, что еще в 70-е годы Пентагон пришел к разумному решению о принятии стандарта на систему команд и архитектуру 16-разрядного микропроцессора для бортовых систем (кстати, стандарт MIL-STD-1750 действует в армии США по сей день). В качестве стандартного языка был избран JOVIAL, поскольку Шварц заложил в него возможность создания кросс-платформенных программ, способных работать в реальном режиме времени.

■ В структуре JOVIAL-машины имеются возможности оптимизации кода для переноса программ с компьютера, на котором ведется разработка, на целевой встроенный компьютер.

Другие языки

- LISP (LISt Processing), изобретенный сотрудником Массачусетского технологического института Джоном Маккарти в 1958 году. Судьба языка LISP феноменальна, он активно использовался в исследованиях, связанных с искусственным интеллектом, вплоть до 90-х годов.
- Потом интерес к нему ослабел, но начиная с 2000 года LISP снова на сцене, сегодня он в первой двадцатке по популярности.
- LISP относят к числу языков, чей индекс популярности в наши дни растет наиболее быстро.

Другие языки

Последним в списке легендарных языков можно назвать Кобол, его история неоднократно описана, он был создан специальным комитетом Минобороны США всего за шесть месяцев. В его основу были положены идеи языков FLOW-MATIC, изобретенного Грейс Хоппер, и COMTRAN Боба Бемера из IBM (последний более известен как один из авторов кода ASCII). Первая спецификация Кобол 60 была опубликована в 1960 году.

РЕФАЛ (РЕкурсивных Функций АЛгоритмический язык), изначально задуманный как метаязык для описания семантики других языков, но впоследствии, с появлением эффективных практических реализаций, он стал использоваться в качестве языка программирования.

TeX (Дональд Кнут) – технология проектирования и разработки ПО одновременно с документированием.

Другие языки

- Декларативное программирование (LISP, РЕФАЛ, TeX).
Аксиоматическая семантика: Хор (1969)
- Логическое программирование: Absys Элкок's (1965), Пролог Colmerauer's (1972)
- Backtracking: Флойд (1967)
- Возможность безопасности: Деннис и Ван Хорн (1965)
- Декларативный параллелизма: Кан (1974)
- Передачи сообщений параллелизма: актер модели Хьюитт (1973)

ООП программирование

Последовавшая затем "кастрация" механизмов **процедурного порождения программ** привела к появлению новых, чисто объектных (как и Smalltalk) языков, тесно интегрированных с современной информационной средой.

Наиболее популярным из них оказался язык Java, ориентированный на кросс платформенную виртуальную машину, адаптированную к функционированию в среде Internet.

Поддерживаемая агрессивной маркетинговой политикой, ОО Java-технология заняла на рынке средств разработки одно из ведущих мест. А последним "писком" является C# - объектно-ориентированный кастрат от Microsoft в пику Java.

ООП программирование

Объектно-ориентированная эпидемия коснулась и языков сценариев, изначально ориентированных на написание небольших скриптов и не нуждающихся в поддержке технологий, связанных с разработкой очень больших программных систем.

ОО "прибамбасы" существуют в Java Script, также и в VB Script.

На высоте оказался идиоматический Perl.

Примечание. Идиома - это не ругательство. Это термин, означающий, что язык нельзя изучить только по формальному описанию и руководству пользователя. Необходимо дополнительно ковыряться в чужом коде и примерах, демонстрирующих особенности использования различных конструкций языка.

ООП программирование

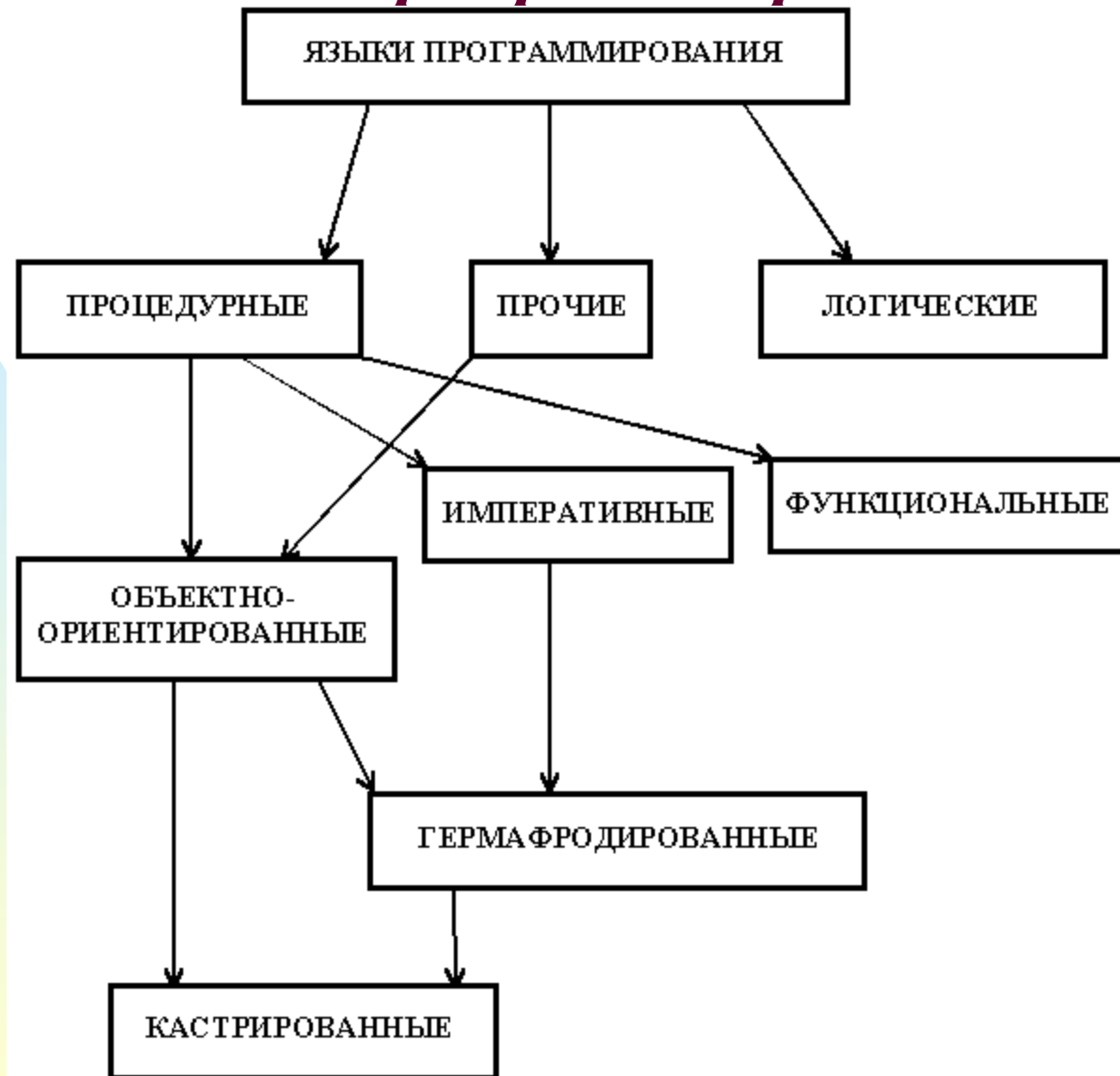


Рис. 1. Генеалогическое дерево объектно-ориентированных языков программирования

Наука информатика

Приходится признать, что главная задача компьютерной науки — «не запутать все до неузнаваемости» — так и не была достигнута.

Увы, большинство наших систем слишком сложны, чтобы не тревожиться об их состоянии, они слишком хаотичны и запутаны, чтобы с ними можно было чувствовать себя уверенно и спокойно.

Обслуживание рядового заказчика в компьютерной отрасли находится на таком низком уровне, что пользователь постоянно ждет, что его система вот-вот рухнет. Мы являемся свидетелями массового, повсеместного распространения полного ошибок программного обеспечения, из-за чего нам должно быть очень стыдно.



Эдсгер В. Дейкстра