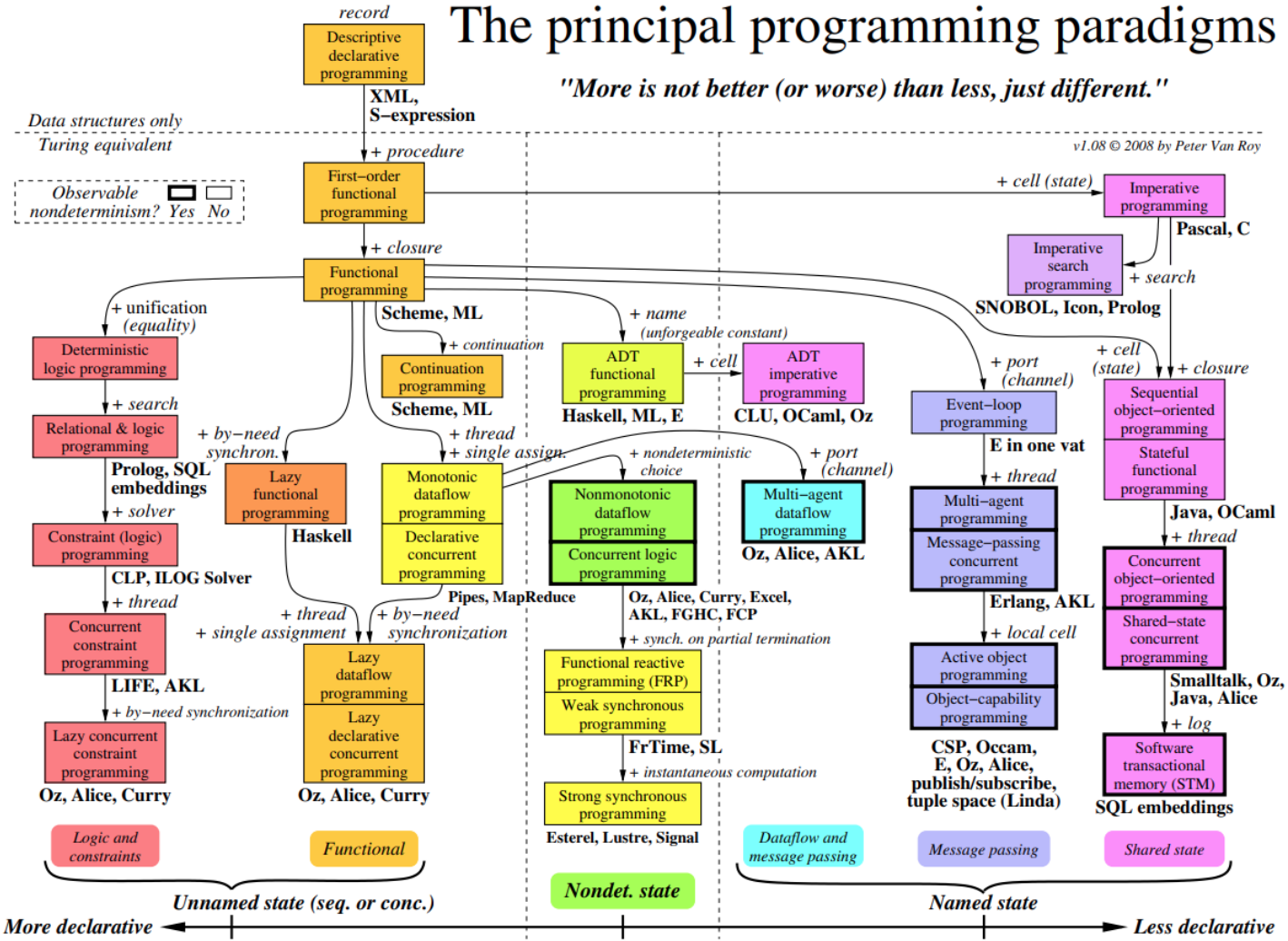
Парадигма программирования

- Парадигма программирования –

это способ классификации языков программирования на основе их особенностей

The principal programming paradigms

"More is not better (or worse) than less, just different."



Парадигма программирования

Императивная - задает последовательность команд, как надо что-то сделать (программа - набор директив, обращенных к ПК) (Fortran, Pascal, Basic, C/C++)

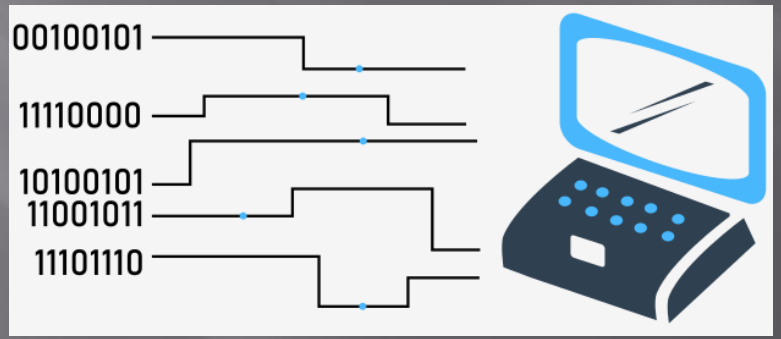
Декларативная - только то, что надо сделать (программа - не набор команд, а описание действий, которые необходимо осуществить (SML, Haskell, Prolog).

Парадигма программирования

“- Безпарадигменнное” программирование“

«Стихийное» - отсутствие технологий, программирование – искусство.

Программы в машинных кодах. Сложность ограничивалась способностью программиста одновременно мысленно отслеживать последовательность выполняемых операций и местонахождение данных при программировании



Парадигма программирования

Ассемблеры - вместо двоичных и 16-ричных кодов используются символические имена данных и мнемоники кодов операций.

В качестве примера приведем программу на языке ассемблера для IBM PC. Программа вычисляет значение a = b + c для целых a, b и c:

```
        .MODEL SMALL
        .DATA
b       DW   5
c       DW   3
a       DW   ?
        .CODE
begin   MOV  AX,@DATA
        MOV  DS,AX
        MOV  AX,B
        ADD  AX,C
        MOV  A,AX
        MOV  AH,4CH
        INT  21H
        END  begin
```

Директива .MODEL задает механизм распределения памяти под данные и команды.
Директива .DATA определяет начало участка программы с данными.
Директивы DW задают типы переменных и их значения.
Директива .CODE определяет начало участка программы с командами.
Команды MOV AX,@DATA и MOV DS,AX записывают адрес сегмента данных в регистр DS (Data Segment).
Для вычисления a используются команды MOV AX, B, ADD AX,C и MOV A,AX.
В директиве END задана метка первой выполняемой программы программы begin.

Перевод программы с языка ассемблера на машинный язык осуществляется специальной программой, которая называется **ассемблером** и является, по сути, простейшим **транслятором**.

Парадигма программирования

Структурное программирование – парадигма программирования, в основе которой лежит представление программы в виде иерархической структуры блоков. В 1960- 1970-х г.г. математически обосновано возможность структурной организации программ (теоремы Бёма-Якопини) и Э.Дейкстра «О вреде оператора goto».

Любую программу можно построить из трёх базовых управляющих конструкций: последовательность, ветвление и цикл; также используются подпрограммы. Это программы без использования оператора goto.

Разработка программы ведётся пошагово, методом «сверху вниз».
В основе - декомпозиция сложных систем с целью последующей реализации в виде отдельных небольших п/п.

Парадигма программирования

Объектно-ориентированное программирование - парадигма разработки, подразумевающая организацию программного кода, ориентируясь на данные и объекты, а не на функции и логические структуры.

Программа – совокупность взаимодействующих объектов.

Каждый О - экземпляр класса.

Классы образуют иерархию с наследованием свойств.

Взаимодействие О - путем передачи сообщений.

Парадигма программирования

-**Функциональное программирование** – парадигма программирования, процесс вычисления трактуется как вычисление значений функций (в математическом понимании) в отличие от функций как п/п в процедурном программировании.
Программа может интерпретироваться как функция с одним или несколькими аргументами. Прозрачное моделирование текста программ математическими средствами (SML)

-**Логическое программирование** – парадигма программирования, основанная на математической логике.
Программа - совокупность логических утверждений (высказываний) и правил вывода. Естественно формализуется логика поведения. Применимы для ЭС, для описаний правил принятия решений (Prolog)

Классификация языков и подходов к программированию

Первые ЯП возникли относительно недавно.
Различные исследователи указывают в качестве времени их создания 20-е, 30-е и даже 40-е годы XX в.
Задача - не установление самого раннего языка, а поиск закономерностей в их развитии.
Первые ЯП, как и первые ЭВМ, были довольно примитивны и ориентированы на численные расчеты. Это были и чисто теоретические научные расчеты (математические и физические), и прикладные задачи, в частности, в области военного дела.
Программы, написанные на ранних ЯП, - линейные последовательности элементарных операций с регистрами, в которых хранились данные.

Классификация языков и подходов к программированию

Ранние ЯП были оптимизированы под аппаратную архитектуру конкретного компьютера, для которого предназначались, и, хотя они обеспечивали высокую эффективность вычислений, до стандартизации было еще далеко.

Программа, которая была вполне работоспособной на одной вычислительной машине, зачастую не могла выполняться на другой.

Таким образом, ранние ЯП существенно зависели от среды вычислений и приблизительно соответствовали современным машинным кодам или языкам ассемблера.

Классификация языков и подходов к программированию

Следующее десятилетие - появление ЯП "высокого уровня", по сравнению с предшественниками - низкоуровневыми языками.

Различие - в повышении эффективности труда разработчиков за счет абстрагирования от конкретных деталей аппаратного обеспечения.

Одна инструкция (оператор) языка высокого уровня соответствовала последовательности из нескольких низкоуровневых инструкций, или команд.

Исходя из того, что программа, по сути, представляла собой набор директив, обращенных к компьютеру, такой подход к программированию получил название ***императивного***.

Классификация языков и подходов к программированию

В императивной (англ. imperative — приказ) парадигме разработчик пишет для компьютера инструкции, которым тот следует. Например:

- сложи два числа;
- если а, то сделай б, а иначе сделай в;
- отправь запрос на сервер;
- открой файл;
- выведи строку.

Другая особенность языков высокого уровня - возможность повторного использования ранее написанных программных блоков, выполняющих те или иные действия, посредством их идентификации и последующего обращения к ним, например, по имени. Такие блоки получили название функций или процедур, и программирование приобрело более упорядоченный характер.

Классификация языков и подходов к программированию

С появлением языков высокого уровня зависимость реализации от аппаратного обеспечения существенно уменьшилась.

Платой за это стало появление специализированных программ, преобразующих инструкции языков в коды той или иной машины, или трансляторов, а также некоторая потеря в скорости вычислений, которая, компенсировалась существенным выигрышем в скорости разработки приложений и унификацией программного кода.

Операторы и ключевые слова новых ЯП были более осмысленными, чем цифровые последовательности кодов, что также обеспечивало повышение производительности труда программистов.

Классификация языков и подходов к программированию

Для обучения новым ЯП требовалось много времени и средств, а эффективность реализации на прежнем аппаратном обеспечении снижалась. Это были временные трудности, и, как показала практика программирования, многие из первых языков высокого уровня оказались настолько удачно реализованными, что активно используются и сегодня.
Один из таких примеров - язык Fortran, реализующий вычислительные алгоритмы.
Другой пример – язык APL, трансформировавшийся в BPL и затем в C. Основные конструкции последнего остаются неизменными уже несколько десятилетий и присутствуют в языке C#.

Примеры других ЯП: ALGOL, COBOL, Pascal, Basic.

Классификация языков и подходов к программированию

В 60-х годах возникает новый подход к программированию, который до сих пор успешно конкурирует с императивным - ***декларативный подход***.

Суть подхода: программа представляет собой не набор команд, а описание действий, которые необходимо осуществить.

Этот подход существенно проще и прозрачнее формализуется математическими средствами.

Следовательно, программы проще проверять на наличие ошибок (тестировать), а также на соответствие заданной технической спецификации (верифицировать).

Высокая степень абстракции также является преимуществом данного подхода. Фактически, программист оперирует не набором инструкций, а абстрактными понятиями, которые могут быть достаточно обобщенными.

Классификация языков и подходов к программированию

На начальном этапе развития декларативным ЯП было сложно конкурировать с императивными в силу объективных трудностей эффективной реализации трансляторов. Программы работали медленнее, однако они могли решать более абстрактные задачи с меньшими трудозатратами.
В частности, язык SML был разработан как средство доказательства теорем.
Различные диалекты языка LISP (Interlisp, Common Lisp, Scheme), возникли потому, что ядро и идеология этого языка оказались эффективными при реализации символьной обработки (анализе текстов).
Другие характерные примеры декларативных языков программирования: SML, Haskell, Prolog.

Классификация языков и подходов к программированию

Одним из путей развития декларативного стиля программирования стал ***функциональный подход***, возникший после создания языка LISP. Его отличительная особенность - любая программа, написанная на таком языке, может интерпретироваться как функция с одним или несколькими аргументами. Такой подход дает возможность прозрачного моделирования текста программ математическими средствами, а значит, интересен с теоретической точки зрения. Сложные программы при таком подходе строятся посредством агрегирования функций. При этом текст программы представляет собой функцию, некоторые аргументы которой можно также рассматривать как функции. Т.о., повторное использование кода сводится к вызову ранее описанной функции, структура которой, в отличие от процедуры императивного языка, математически прозрачна.

Классификация языков и подходов к программированию

Более того, типы отдельных функций, используемых в функциональных языках, м.б. переменными. Т.о. обеспечивается возможность обработки разнородных данных (упорядочение элементов списка по возрастанию для целых чисел, отдельных символов и строк) или полиморфизм.
Другое важное преимущество реализации языков функционального программирования - автоматизированное динамическое распределение памяти компьютера для хранения данных. Программист избавляется от обязанности контролировать данные, а при необходимости может запустить функцию "сборки мусора" – очистки памяти от тех данных, которые больше не потребуются программе (обычно этот процесс периодически инициируется компьютером).

Классификация языков и подходов к программированию

Таким образом, при создании программ на функциональных языках программист сосредотачивается на области исследований (предметной области) и в меньшей степени заботится о рутинных операциях (обеспечении правильного с точки зрения компьютера представления данных, "сборке мусора" и т.д.).
Поскольку функция является естественным формализмом для языков функционального программирования, реализация различных аспектов программирования, связанных с функциями, существенно упрощается. В частности, становится прозрачным написание рекурсивных функций, т.е. функций, вызывающих самих себя в качестве аргумента. Естественной становится и реализация обработки рекурсивных структур данных (списков – базовых элементов для языков семейства LISP, деревьев и др.)

Классификация языков и подходов к программированию

Благодаря реализации механизма сопоставления с образцом, такие языки как ML и Haskell вполне применимы для символьной обработки. Недостатки: нелинейная структура программы, относительно невысокая эффективность реализации. Первый недостаток достаточно субъективен, а второй успешно преодолен современными реализациями, в частности, рядом последних трансляторов языка SML.

В 70-х годах возникла еще одна ветвь языков декларативного программирования, связанная с проектами в области искусственного интеллекта, а именно языки ***логического программирования:*** программа представляет собой совокупность правил или логических высказываний, допустимы логические причинно-следственные связи.

Классификация языков и подходов к программированию

Языки логического программирования базируются на классической логике и применимы для систем логического вывода, в частности, для экспертных систем. На языках логического программирования естественно формализуется логика поведения, и они применимы для описаний правил принятия решений, например, в системах, ориентированных на поддержку бизнеса.
Важным преимуществом такого подхода является достаточно высокий уровень машинной независимости, а также возможность откатов – возвращения к предыдущей подцели при отрицательном результате анализе одного из вариантов в процессе поиска решения (например, очередного хода при игре в шахматы), что избавляет от необходимости поиска решения путем полного перебора вариантов и увеличивает эффективность реализации.

Классификация языков и подходов к программированию

Один из недостатков логического подхода в концептуальном плане - специфичность класса решаемых задач. Другой недостаток практического характера - сложность эффективной реализации для принятия решений в реальном времени, например, для систем жизнеобеспечения.
Нелинейность структуры программы является особенностью декларативного подхода и представляет собой оригинальную особенность, а не объективный недостаток.
Примеры языков логического программирования: Prolog (PROgramming in LOGic) и Mercury.

Классификация языков и подходов к программированию

Важным шагом на пути к совершенствованию языков программирования стало появление ***объектно-ориентированного подхода*** к программированию (ООП) и соответствующего класса языков.

Программа представляет собой описание объектов, их свойств (атрибутов), совокупностей (классов), отношений между ними, способов их взаимодействия и операций над объектами (методов). Преимущество подхода - концептуальная близость к предметной области произвольной структуры и назначения. Механизм наследования атрибутов и методов позволяет строить производные понятия на основе базовых и таким образом создавать модель сколь угодно сложной предметной области с заданными свойствами.

Классификация языков и подходов к программированию

Другим важным свойством ООП является поддержка механизма обработки событий, которые изменяют атрибуты объектов и моделируют их взаимодействие в предметной области.
Перемещаясь по иерархии классов от более общих понятий предметной области к более конкретным (или от более сложных – к более простым) и наоборот, программист получает возможность изменять степень абстрактности или конкретности взгляда на моделируемый им реальный мир.
Использование ранее разработанных (возможно, другими коллективами программистов) библиотек объектов и методов позволяет значительно сэкономить трудозатраты при производстве ПО, в особенности типичного.

Классификация языков и подходов к программированию

Объекты, классы и методы могут быть полиморфными, что делает реализованное ПО более гибким и универсальным.
Сложность адекватной (непротиворечивой и полной) формализации объектной теории порождает трудности тестирования и верификации созданного ПО. Это обстоятельство является одним из самых существенных недостатков ООП к программированию.

Наиболее известный пример ЯП ООП - язык C++, резвившийся из императивного языка C. Его прямым потомком и логическим продолжением является язык C#. Другие примеры: Visual Basic, Eiffel, Oberon.

Классификация языков и подходов к программированию

Развитием событийно управляемой концепции ООП стало появление в 90-х годах целого класса языков программирования, которые получили название ***языков сценариев или скриптов***.

В рамках данного подхода программа представляет собой совокупность возможных сценариев обработки данных, выбор которых инициируется наступлением того или иного события (щелчок по кнопке мыши, попадание курсора в определенную позицию, изменение атрибутов того или иного объекта, переполнение буфера памяти и т.д.). События могут инициироваться как операционной системой (в частности, Microsoft Windows), так и пользователем.

Основные достоинства языков данного класса унаследованы от ОО ЯП. Это интуитивная ясность описаний, близость к предметной области, высокая степень абстракции, хорошая переносимость.

Классификация языков и подходов к программированию

Широкие возможности повторного использования кода также унаследованы сценарными языками от объектно-ориентированных предков.
Существенным преимуществом языков сценариев является их совместимость с инструментальными средствами автоматизированного проектирования и быстрой реализации ПО - CASE- (Computer-Aided Software Engineering) и RAD- (Rapid Application Development) средствами.
Одним из наиболее передовых инструментальных комплексов, предназначенных для быстрой разработки приложений, является Microsoft Visual Studio .NET.

Классификация языков и подходов к программированию

Вместе с достоинствами ОО подхода языки сценариев унаследовали и ряд недостатков: сложность тестирования и верификации программ и возможности возникновения в ходе эксплуатации множественных побочных эффектов, проявляющихся за счет сложной природы взаимодействия объектов и среды, представленной интерфейсами с большим количеством одновременно работающих пользователей программного обеспечения, операционной системой и внешними источниками данных.
Примеры сценарных языков программирования: VBScript, PowerScript, LotusScript, JavaScript.

Классификация языков и подходов к программированию

Важный класс ЯП – ***языки поддержки параллельных вычислений*.** Программы, написанные на этих языках, представляют собой совокупность описаний процессов, которые могут выполняться как в действительности одновременно, так и в псевдопараллельном режиме. В последнем случае устройство, обрабатывающее процессы, функционирует в режиме разделения времени, выделяя время на обработку данных, поступающих от процессов, по мере необходимости (а также с учетом последовательности или приоритетности выполнения операций).

Языки параллельных вычислений позволяют достичь заметного выигрыша при обработке больших массивов информации, поступающих от одновременно работающих пользователей, либо имеющих высокую интенсивность (как, например, видеоинформация или звуковые данные высокого качества).

Классификация языков и подходов к программированию

Другой значимой областью применения языков параллельных вычислений являются системы реального времени, в которых пользователю необходимо получить ответ от системы непосредственно после запроса. Такого рода системы отвечают за жизнеобеспечение и принятие ответственных решений.

Недостатки: высокая стоимость разработки ПО, следовательно, создание относительно небольших программ широкого (например, бытового) применения зачастую нерентабельно.

Примеры ЯП с поддержкой параллельных вычислений: Ada, Modula-2 и Oz.

Приведенная классификация не является единственно верной, поскольку ЯП постоянно развиваются и совершенствуются, и недавние недостатки устраняются с появлением необходимых инструментальных средств и теоретических обоснований.

Классификация языков и подходов к программированию

Итоги. Подходы к программированию:
•ранние неструктурные подходы;
•структурный или модульный подход (задача разбивается на подзадачи, затем на алгоритмы, составляются их структурные схемы и осуществляется реализация);
•функциональный подход;
•логический подход;
•объектно-ориентированный подход;
•смешанный подход (некоторые подходы можно комбинировать);
•компонентно-ориентированный (программный проект рассматривается как множество компонент, такой подход принят, в частности, в .NET);
•чисто объектный подход (идеальный с математической точки зрения вариант, который пока не реализован практически).