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.

Наука информатика

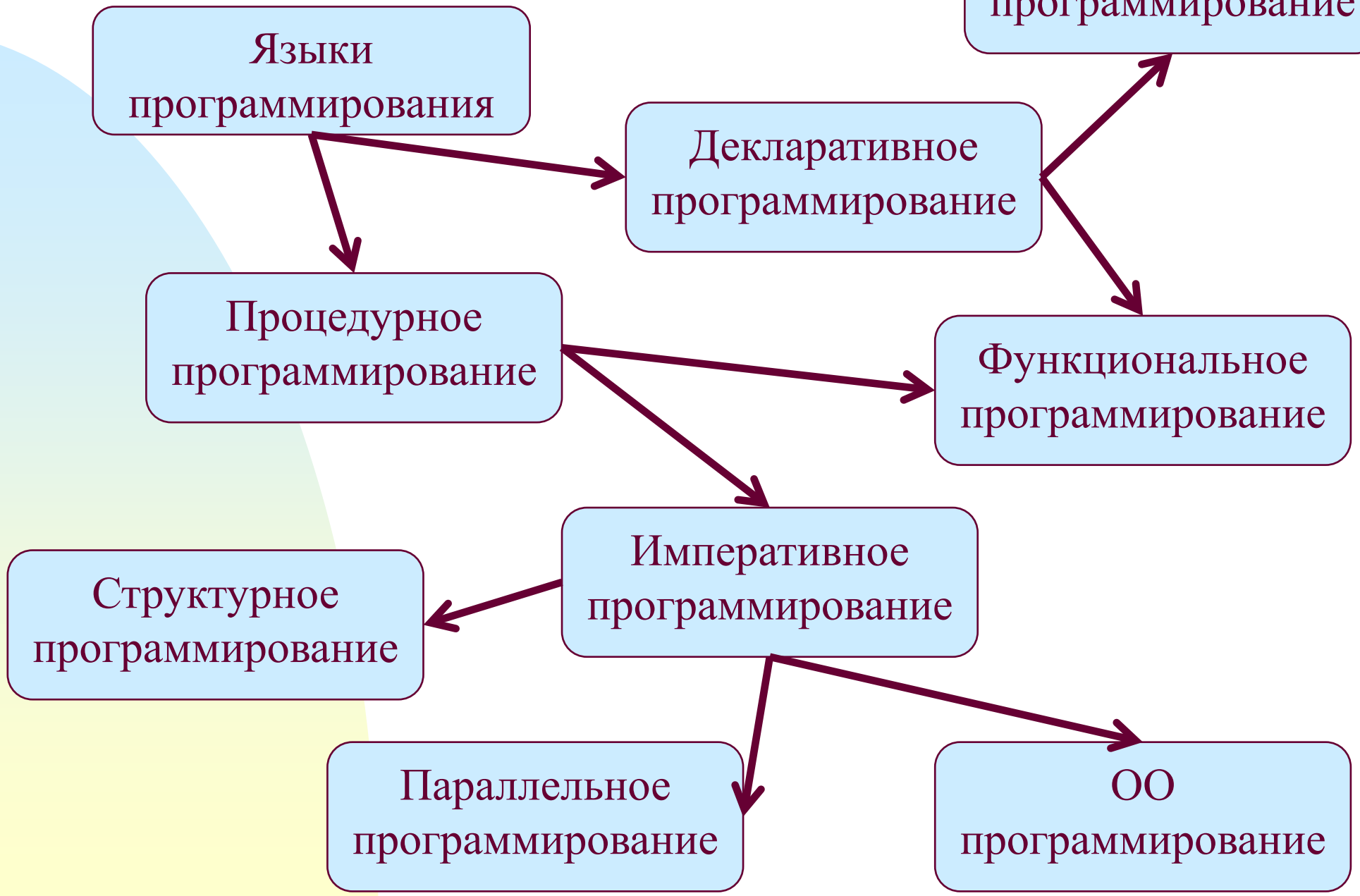
Приходится признать, что главная задача компьютерной науки — «не запутать все до неузнаваемости» — так и не была достигнута.

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

Программирование



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.

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.

Парадигмы программирования

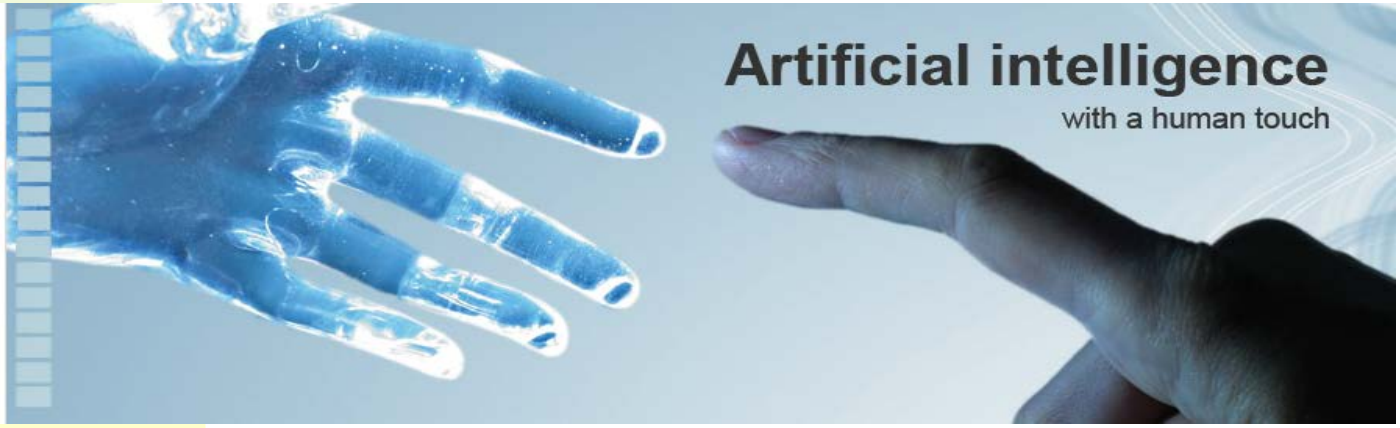
Популярность ООМ привела к упадку других технологий программирования (особенно декларативного), зачастую даже прямо не связанных с методами разработки программного обеспечения.

Наибольшего расцвета в "золотой век" процедурного подхода достигли структурное и *функциональное* программирование.

Динамично развивалось параллельное программирование, различные ответвления которого прекрасно согласовывались как с идеологией процедурных языков, так и с множеством архитектур параллельных вычислительных систем.

Области применения декларативных языков

- Создание систем искусственного интеллекта
- Автоматическое доказательство теорем
- Разработка экспертных систем и оболочек экспертных систем
- Создание систем поддержки принятия решений
- Разработка систем обработки естественного языка
- Построение планов действий роботов



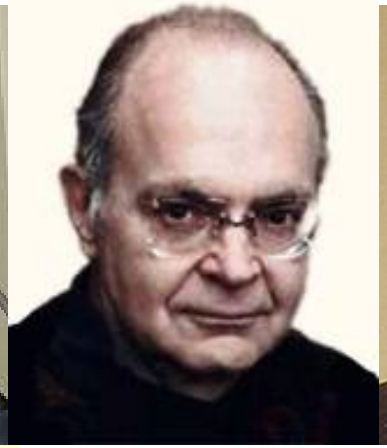
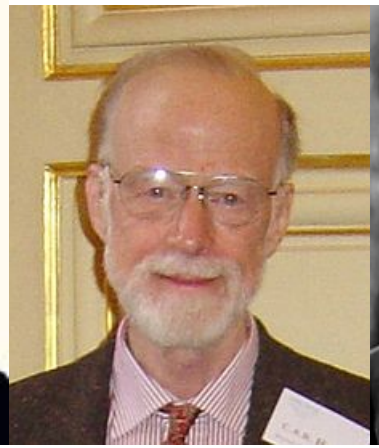
Преимущества логического программирования

- Задача программиста – описание логической модели предметной области в терминах объектов, их свойств и отношений между ними (без деталей): описание данных и логики их обработки ~ аналогия с ООП
- Удобство описания отношений между объектами (реляционная модель)
- Компактность кода (обработка структурированных данных, лог. правила)
- Возможность перебора и поиска различных решений, заложенная в язык
- Легкость понимания (описание отд. правил), отладки программ (trace)
- Легкость описания сложных структур данных (деревья, списки и т.п.)
- Эффективный метод вычислений – рекурсия
- Отсутствие указателей, операторов присваивания и безусловного перехода
- Множество областей для применения: автоматический перевод, обработка текстов, экспертные системы, САПР, Data-minig системы, автоматическое управление, СУБД, символьные вычисления

Структурное программирование

Методология структурного императивного программирования — подход, заключающийся в задании хорошей топологии императивных программ, в том числе отказе от использования глобальных данных и оператора безусловного перехода, разработке модулей с сильной связностью и обеспечении их независимости от других модулей.

Структурное программирование — методология разработки программного обеспечения, в основе которой лежит представление программы в виде иерархической структуры блоков. Предложена в 70-х годах XX века Э.Дейкстрой, разработана и дополнена Н. Виртом, Д.Кнутом, Э.Хоаром, Р.Флойдом и Х.Миллсом.

Структурное программирование

- Любая программа представляет собой структуру, построенную из трёх типов базовых конструкций:
  - **последовательное исполнение** — однократное выполнение операций в том порядке, в котором они записаны в тексте программы;
  - **ветвление** — однократное выполнение одной из двух или более операций, в зависимости от выполнения некоторого заданного условия;
  - **цикл** — многократное исполнение одной и той же операции до тех пор, пока выполняется некоторое заданное условие (условие продолжения цикла).

В программе базовые конструкции могут быть вложены друг в друга произвольным образом, но никаких других средств управления последовательностью выполнения операций не предусматривается.

Структурное программирование

- **Теорема о структурном программировании:** ***Теорема Бома-Якопини (1966 год).***

- В основе структурного программирования лежит теорема, которая доказана в теории программирования.
- Суть ее в том, что алгоритм для решения любой логической задачи можно составить только из структур «следование, ветвление, цикл».
- Их называют базовыми алгоритмическими структурами.

Упадок структурного программирования

- Самый ощутимый удар был нанесен по структурному программированию (являющемуся в свое время наиболее популярным методом разработки программ).
- Разработка кода, основанного на функциональных зависимостях решаемой задачи (удобочитаемые императивные конструкции).
- Отличаясь в первоначальном варианте от предшествующих методов лишь запретом на использование оператора безусловного перехода, оно в дальнейшем дало толчок разнообразным подходам, сочетающим функциональную декомпозицию с декомпозицией данных и использованием абстрактных типов данных.

Упадок структурного программирования

- Появились отдельные методики разработки структур данных и средств управления программой, а также методы обеспечивающие формирования кода по разработанным структурам данных. Ясная логическая структура управляющих примитивов привела к созданию методов доказательства правильности программ [Дейкстра].

- Часть достигнутых результатов была использована в методах структурного анализа и проектирования, применяемых до сих пор.

Упадок структурного программирования

- Исходя из того, что структура обрабатываемых данных в основном представляла иерархические, иногда рекурсивные, и-или деревья, процедуры обработки соответствующих структур строились на основе вложенных условных операторов или переключателей.
- Именно эти конструкции и оказались уязвимым местом структурных программ. Любое изменение данных вело к изменению их обработчиков. Приходилось модифицировать практически каждую процедуру, связанную с обработкой соответствующих структур.
- Если для простых программ внесение таких изменений не является трудоемкой задачей, то в сложных промышленных продуктах, насчитывающих сотни тысяч строк кода, немудрено получить дополнительные ошибки.

Упадок структурного программирования

- Наступление ООП как раз и было связано с отказом от представления вариантных структур. Вместо этого, альтернативные понятия выстраивались путем подключения объектов, наследующих от базового класса.

- Встраивание обработчика внутрь класса и возможность его переопределения в наследниках позволили локализовать проблему. Теперь любые изменения структуры данных определяли изменения только тех классов, в которых находились эти данные. Разработка ОО методологий и эффективных инструментов ОО программирования позволили ускорить процесс разработки более надежных и расширяемых программ, пригодных к повторному использованию.

- Моделирование подобных объектов при структурном подходе, как и любое другое моделирование, не помогало улучшить ситуацию, так как обнаружение ошибок с фазы компиляции переносилось на период исполнения программы.

Упадок функционального программирования

- Функциональное программирование тоже опирается на процедурный подход.
- Его основной спецификой является то, что функции обмениваются между собой данными непосредственно, то есть, без использования промежуточных переменных.
- Другой особенностью является использование динамической типизации, суть которой заключается в формировании типов данных вместе с их значениями в ходе вычислений.
- Это обеспечивает полиморфизм аргументов функций, и позволяет вводить в них любые структуры данных.

Упадок функционального программирования

- Однако дальнейшие вычисления внутри функции требуют проверки дополнительных условий. Например, необходимо отличать списки от атомов. Для списков часто требуется знать количество аргументов, их внутреннюю структуру и т.д.
- Поэтому, по организации ветвлений в ходе обработки различных данных функциональное программирование почти не отличается от императивного.
- Меньшая распространенность, по сравнению с процедурными языками, низкая эффективность инструментальных средств и ряд других причин привели к тому, что в настоящее время функциональное программирование применяется редко за исключением АСУТП.

Преимущества функционального программирования

- Программа представляет собой множество вычисляемых функций
- Отсутствует операция присваивания (осн. Императивный оператор)
- Порядок выполнения программы (вычислений) несущественен
- Быстродействие (по сравнению с аналогичными императивными вычислениями)
- Компактность кода (особенно для списков и вариантных типов)
- Суперпозиции/склеивания функций: из простых – сложные, редукция
- Суперпозиция моделей: склеивание программ, ленивые вычисления
- Численное дифференцирование/интегрирование (в т.ч. модульное)
- Алгоритмы искусственного интеллекта (эвристический поиск и т.п.)
- Распараллеливание вычислений (Haskell)
- Интерактивная отладка, Unit-тестирование, карринг
- Сопоставление с образцом

Упадок параллельного программирования

- Широкое распространение ООП сильно ударило и по параллельному программированию, не смотря на отсутствие видимой связи.
- Основная причина этой ситуации лежит в том, что, по своей сути, параллельное программирование - это взаимодействие процессов.
- Внешнее проявление любого процесса - это его функциональная или процедурная оболочка. Все что касается управления процессами: задержки, ожидания, синхронизация, распараллеливание и т.д., лежит внутри этой оболочки.

Упадок параллельного программирования

- Такое соответствие позволило разработать и успешно применять разнообразные языки параллельного программирования на основе уже существующих процедурных и функциональных языков.
- Возможности распараллеливания никак не влияли на логику управления вычислениями, зависящую от структур данных. Параллелизм только позволял ускорить выполнение операций, не зависящих информационно друг от друга.

- Можно также отметить, что в это же время получила широкое развитие теория параллельного программирования. Были разработаны разнообразные модели параллельных вычислений, поддерживающие распараллеливание от уровня процессов-программ до уровня команд.
- Именно теория позволила создать нетрадиционные языки параллельного программирования, ряд из которых нашли практическое применение.

Упадок параллельного программирования

- Вместе с тем, популярность и распространенность языков параллельного программирования сильно зависели от текущего положения дел на рынке архитектур параллельных вычислительных систем.

- Появление матричных процессоров подняло интерес к матричному и векторному программированию с применением распараллеливания на уровне массивов (массивный параллелизм). Взлет векторных ЭВМ, использующих конвейерную обработку данных, привел к созданию и популяризации векторных языков и языков конвейерного программирования.

Упадок параллельного программирования

- Дальнейшее развитие параллельной обработки данных было непосредственно связано с развитием микропроцессорной техники, что привело к созданию мультимикропроцессорных систем на основе распределенной и общей памяти. Популярность получили языки, обеспечивающие распараллеливание на уровне ветвей команд. С течением времени только уменьшалась длина этой ветви, начиная от процесса и заканчивая потоком (нитью).

- Однако, параллелизм на уровне команд до сих пор не реализован (исключение машина Мельникова). Отсутствует технологическая база для разработки соответствующих мультипроцессорных архитектур.

Упадок параллельного программирования

- В настоящее время языки параллельного программирования практически перестали использоваться для разработки больших программ. Как и в ранее описанных случаях, их сгубила процедурность.
- Естественно, что современные ОО языки поддерживают параллельную обработку данных, которая осуществляется либо через системные вызовы ОС, либо непосредственно встроена в ОО язык (например, в Java). Эта поддержка не реализует какую-либо оригинальную модель вычислений. Она просто воспроизводит возможности ОС. Последние же, обычно, поддерживают те возможности, которые предоставляются аппаратурой.
- Поэтому, можно сказать, что ООП затормозило практическое развитие методов параллельной обработки данных.

Технологии программирования Баз Данных (Информационных Систем) – что такое ОБЪЕКТ?

На сегодняшний день *в программировании* не существует точной формулировки понятия *объект*!

Точнее сказать, их столько много, что нет никакой возможности выбрать правильную формулировку.

Диапазон широк, от общефилософской "Все есть объект" (значит и стул, на котором сидит программист, тоже объект программирования), до слишком ограничивающих формулировок, например, в ООП объект наделяют обязательным свойством наследования. Некоторые формулировки просто устарели, хотя находятся в постоянном употреблении, например, утверждение о том, что объект должен находиться в оперативной памяти компьютера (тогда как быть с объектами, расположенными в БД?).

Парадигмы программирования

Что такое ОБЪЕКТ:

Программным объектом *называется* ***сущность,*** *находящаяся в вычислительной среде, образованная* ***путем преобразования программного кода.***

Программный объект может обеспечивать доступ к своим функциональным возможностям и ресурсам со стороны других программных объектов.

ООП программирование

Изменения в представлениях программистов вызваны с появлением сетевых технологий, WWW, WWW программирования и идеологии распределенных информационно-вычислительных систем (GRID).

ПРОЦЕССЫ на смену **ОБЪЕКТОВ**

Парадигмы программирования

Казалось бы, что такого в ООП? Ну, свалили в одну упаковку данные и процедуры, а потом начали обертывать этой "бумажкой" экземпляры коробок и использовать для доступа к содержимому название коробки.

Ведь почти то же самое можно сделать и с процедурой. Запихнуть в нее другие процедуры и данные. Однако, существует один маленький нюанс: процедура после завершения ее вызова теряет свое состояние (за исключением статических переменных, которые не могут относиться к разным экземплярам), а экземпляр класса (объект) продолжает существовать в пассивном состоянии, периодически нарушаемом вызовами его методов.

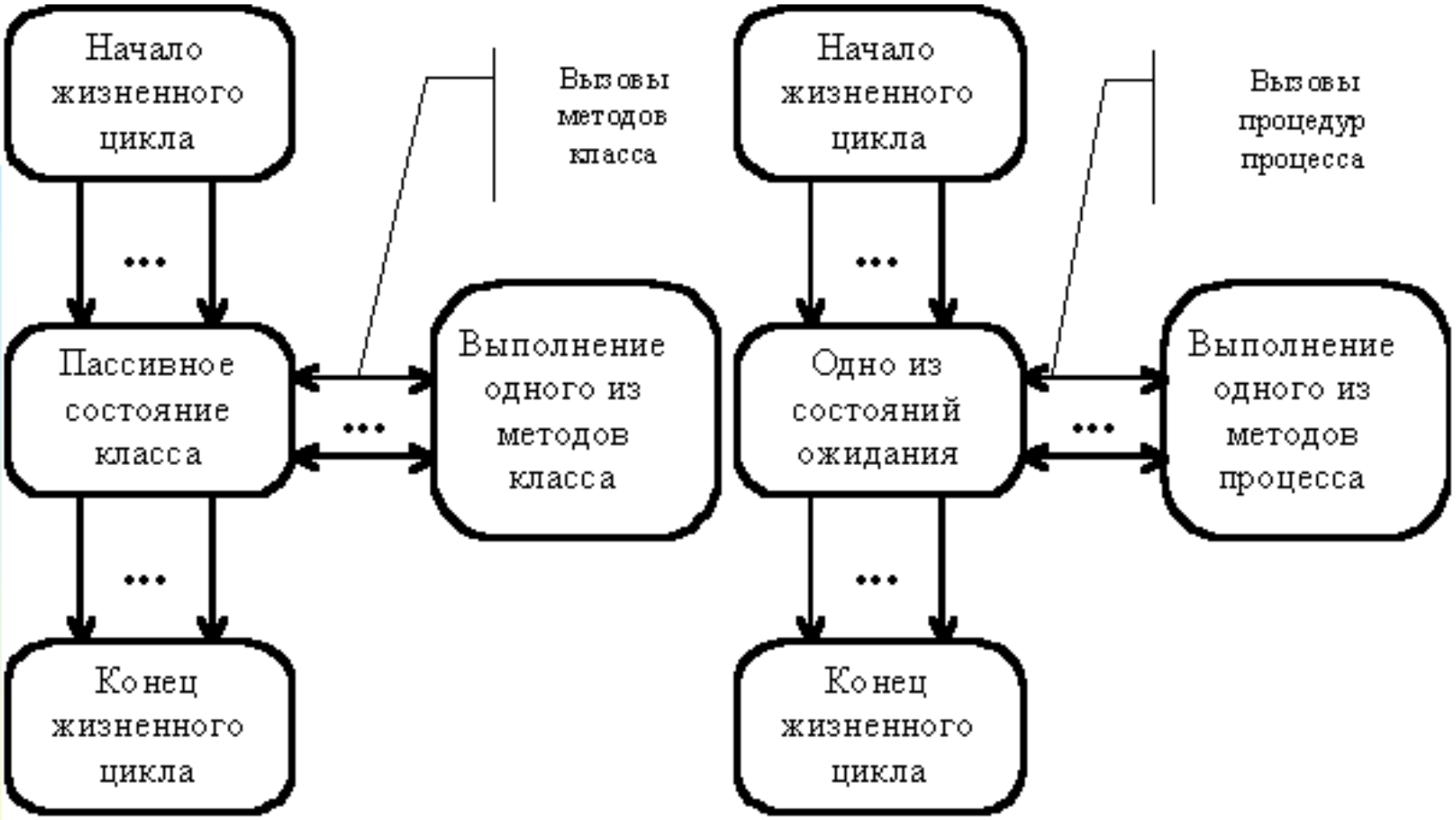
Парадигмы программирования

Но, процедура - это частный случай процесса, который, наряду с выполнением операций может переходить в состояние ожидания. Сравним жизненные циклы класса и элементарного процесса.

Их практическая эквивалентность говорит о том, что процесс может делать все то же, что и класс.

А уж сымитировать с его помощью модель класса - вообще нет проблем. При этом процесс, в общем случае, обеспечивает такую динамику поведения, которая не "снилась" ни одному другому программному объекту (достаточно почитать классиков [Хоар]).

Парадигмы программирования



а) жизненный цикл класса

б) жизненный цикл процесса

Рис. 2. Модели жизненных циклов класса и процесса

Парадигмы программирования

Такая эквивалентность позволяет говорить о том, что языки, ориентированные на процессы, могут успешно использоваться для написания программ в объектно-ориентированном стиле.

Надо только расширить процессы механизмами, используемыми в ОО языках программирования. Например, без особых проблем можно разработать язык P++, обеспечивающий поддержку процессо-ориентированного программирования (ПОП) в объектно-ориентированном стиле.

Не вдаваясь в особенности синтаксиса и семантики этого языка, рассмотри пример фрагментов кода (кто хоть немного знает C++, тот поймет: Пример кода на PP.pdf

Парадигмы программирования

- Возникает вопрос: во что выльется техническая реализация таких процессов и возможна ли она?

- Начнем с того, что создание и поддержка процессов осуществляется достаточно давно и различными способами. Практически все операционные системы в настоящее время поддерживают мультипрограммный и мультипроцессорный режимы, как на уровне процессов, так и на уровне ветвей (потоков, нитей). Поэтому, вряд ли могут возникнуть проблемы по порождению процессов только из-за того, что они описаны в стиле, напоминающем объектный. Более того, поддержка процессов и задач осуществляется даже на аппаратном уровне практически во всех современных микропроцессорах.

Парадигмы программирования

- Всем нам известные 32-х разрядные процессоры для ПК фирмы Intel имеют такую возможность с момента своего появления. Глобальность этого механизма, сочетание виртуальной и страничной адресации позволяют использовать все возможности современных операционных систем.

- Можно возразить, что применение процессов вместо классов значительно увеличит объем используемой оперативной памяти. А кто сказал, что эти расходы увеличатся намного? А кто подсчитал, что расходы памяти увеличатся? А кто вообще сейчас считает память?

Парадигмы программирования

- Традиционно процессы организованы так, что обеспечивают запуск из одной точки с одним вариантом начальных значений. Альтернативные ветви, выходящие из точки ожидания обычно порождаются путем выполнения условных операторов.
- Поэтому, могут появиться сомнения по поводу наличия нескольких различных точек запуска процессов, эквивалентных конструкторам класса. Червь сомнения может глодать нас и при обсуждении возможности наличия у процесса внутренних методов, отвечающих за отдельные ветви, эквивалентные методам класса.
- Перефразируя одну печально известную фразу можно сказать: "Оставь сомнения всяк, в процесс входящий"!

Парадигмы программирования

- Множество точек входов и ранее встречалось, даже в процедурах. Достаточно вспомнить PL/1 или Фортран. А реализация оболочки процесса может быть такой же, как и у компонента используемого в COM.

- Эта двоичная оболочка, фиксирующая только указатель на таблицу функций, позволяет интерпретировать себя как угодно. Кстати, большинство реализаций класса используют точно такую же структуру.

- Поэтому, вряд ли у процессов возникнут проблемы с инкапсуляцией, полиморфизмом и наследованием.

Парадигмы программирования

- Итак, технических проблем вроде бы не видно. Но остаются морально-этические. Спрашивается: "А зачем все это надо"? Разве не достаточно, для полного счастья, объектно-ориентированного подхода? Конечно, можно было бы сказать, что это месть за процедурное и параллельное программирование.

- Однако вряд ли такой аргумент можно считать убедительным. Скорее всего, это попытка показать, что после объединения процедур и данных в единую оболочку не существует однозначного толкования и интерпретации возникшего винегрета.

Парадигмы программирования

- На уровне реализации это может объект, а может - процесс. Поэтому, не стоит серьезно относиться и к словам Гради Буча о том, что объектная декомпозиция имеет несколько чрезвычайно важных преимуществ перед алгоритмической.

- Алгоритмическая декомпозиция позволяет учитывать модель состояний, если опирается на декомпозицию процессов. И в перспективе она может использовать инкапсуляцию, наследование и полиморфизм.

- Поэтому, вряд ли стоит ставить крест на структурном анализе и проектировании, начинающемся с описания бизнес процессов.

Парадигмы программирования

- Следует обратить внимание и на потенциальные преимущества процессов перед объектами. Они, сами по себе, поддерживают параллелизм решаемой задачи, а также позволяют задавать несколько точек ожидания вместо одного пассивного состояния.
- С каждой из таких точек можно связать свой интерфейс, обеспечив, тем самым дополнительные возможности. Вообще, понятие процесса гораздо шире, чем класса, а это позволяет говорить о неисповедимых путях дальнейшее развития данного направления технологии программирования.

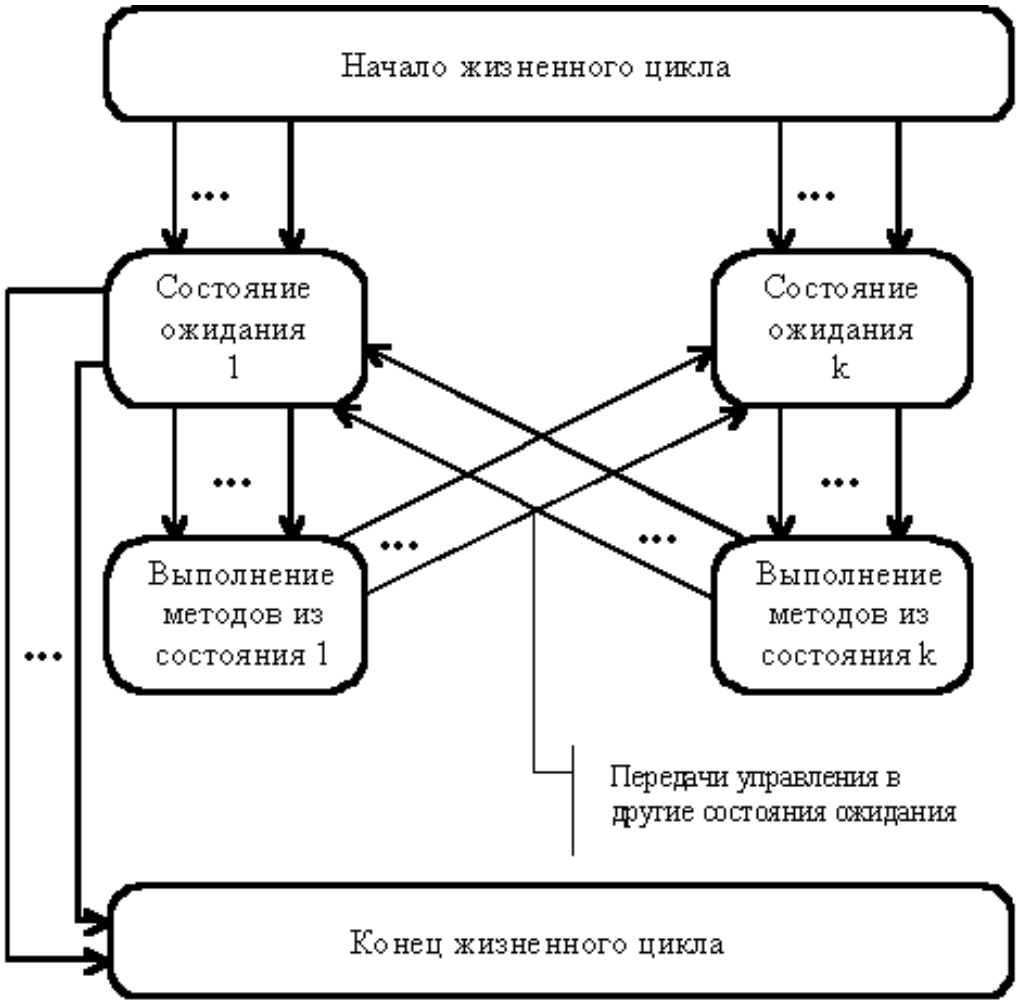


Рис. 3. Более детальная модель жизненного цикла процесса

Парадигмы программирования

- Описывая основные достоинства ООМ Гради Буч совершенно серьезно говорит, что: "Объектная модель позволяет в полной мере использовать выразительные возможности объектных и объектно-ориентированных языков программирования".
- Как будто построение объектной модели происходило не на основе этих самых языков! Ведь любая методология разрабатывается с использованием уже накопленных в предметной области эмпирических фактов и практических результатов, коими, в данном случае, и являлись существующие языки программирования. Ведь до появления ООМ прошел не один десяток лет, если плясать от Симулы-67.
- Только *идиоты*, политики и менеджеры могут создавать методологии, которые не используют выразительные возможности языков. Вряд ли можно отнести Буча к первым двум категориям. Это голова и гигант мысли в одном флаконе! Следовательно, он немного перестарался, чтобы раскрутить свои методы.
- Хотя, и политики в программировании более чем достаточно.

Парадигмы программирования

- ОО подход породил множество постулатов и догм, определяющих правильный стиль программирования. Со временем они перекочевали в различные книжки.
- Ряд зафиксированных приемов оказались связанными со спецификой конкретных языков программирования. Другие были заявлены как универсальные и обосновались в "приемах объектно-ориентированного проектирования".
- Не отрицаю полезность накопленного эмпирического опыта и с удовольствием использую его в своей практике. Но, вместе с тем, создается впечатление, что мы пользуемся не той системой отсчета. Что-то напоминает геоцентрическую систему Птолемея, которая поставила землю в центр вселенной. Для простейших расчетов все идет нормально. Но как только переходим к более сложным и точным расчетам, каждая планета начинает двигаться по каким-то непонятным, дополнительным окружностям. И без уточняющих правил, исключений, измерительных инструментов, а также шаманских ритуальных плясок не обойтись.
- Я не думаю, что в мгновение ока поставлю в центр нашей системы Солнце. Скорее всего, у меня на этом месте расположится Марс.

Парадигмы программирования

- Центром вселенной в мире ООП являются интерфейсы, точнее сигнатуры процедур и функций. Именно через сигнатуры должно идти все взаимодействие классов. В противном случае нарушается инкапсуляция, затрудняется модификация, а также возникает множество других проблем. Нельзя напрямую обратиться к переменной экземпляра класса. Только через методы интерфейса.
- Все это происходит оттого, что между клиентом, использующим класс и состоянием класса должно обязательно стоять промежуточное звено, служащее посредником при передаче данных. Роль этого звена обычно исполняет тело процедуры, которое и решает проблемы взаимодействия сигнатуры с данными класса. Даже в том случае, когда необходимо осуществлять только прием или передачу простейших данных, приходится пользоваться методом типа GetSetPut. Вместе с тем, метафора непосредственно доступной переменной часто бывает полезной и облегчает программирование. Поэтому, в ряде ОО языков использование методов скрывается под таким понятием как свойство. Однако и в этой ситуации реализация остается прежней.

Парадигмы программирования

- Однако косвенное взаимодействие необязательно должно обеспечиваться через явный вызов процедуры. Еще на заре параллельного программирования был придуман обмен данными между процессами через почтовые ящики. Одни процессы загружали почту (информацию), а другие забирали ее.
- Внутренняя организация процессов оставалась при этом закрытой. Достаточно было только договориться о формате обмена данными и подключиться к почтовым ящикам. При этом обмен мог вестись через простые переменные очереди, стеки и любые другие объекты одинаковым для процессов способом.
- Любые внутренние изменения данных в процессах никоим образом не влияли на их взаимодействие.

Парадигмы программирования

- Так что, интерфейс - это не только процедуры. Он может быть выстроен и на основе модели данных. При этом неважно, что посредники между почтовыми ящиками и процессами являются процедурами, так как нас совершенно не волнуют ни их сигнатуры, ни внутренние модификации процессов.
- Можно легко менять процедуры, обслуживающие почтовые ящики. Можно даже использовать несколько разных процедур процесса для обслуживания одного ящика, выбирая нужную процедуру в зависимости от текущего состояния ожидания.

Выводы

- Существуют императивный и декларативный стили программирования (и соответствующие им языки).
- Языки программирования принято разделять по уровням (низкий, высокий)
- Существует множество парадигм программирования, основными из которых являются: Процедурная, Функциональная, Логическая, Объектно-ориентированная (а также ряд узкоспециализированных парадигм)
- Существует ряд задач, для которых удобно использовать языки функционального и/или логического программирования, достоинствами которых являются компактность программного кода и высокая скорость выполнения программ.