Профессиональное программирование Системно-философский подход*

Программирование – это искусство создавать программные продукты, которые написаны на языке программирования.

Основы профессионального программирования представлены тремя группами средств.

Общесистемная группа

Представленные в ней методы и подходы определяют основную идеологию и принципы программного проекта. Группа состоит из двух направлений:

- **Методология программирования – совокупность методов, применяемых в жизненном цикле программного обеспечения и объединенных общим философским подходом.**
- **Технология программирования – это технологические процессы и порядок их прохождения – стадии (с использованием знаний, методов и средств).**

Инструментальная группа

- **Языки программирования – это формальная знаковая система, которая предназначена для написания программ, понятной для компьютера.**

Системы программирования – это совокупность языка программирования и инструментальной программной оболочки, представляющая интегрированную среду, которая может содержать:
компилятор; редактор; средства компоновки, отладки и загрузки программы; библиотеку стандартных программ; справочную службу; средства поддержки коллективной разработки.

Базисная группа

Эта группа включает два направления, представляющих собой основные платформы, на которых базируется программный проект.

- **Архитектурная платформа – это основные архитектуры набора команд; организация вычислительной системы (процессор, память, шина, внешние устройства); параллельные и распределенные архитектуры.**
- **Операционная платформа – это различные механизмы и структуры операционных систем, управляющих функционированием вычислительной системы (компьютера).**

* **На основе книги: Одинцов И. О. Профессиональное программирование. Системный подход.**

Профессиональное программирование

Методология программирования

Методология – это совокупность методов, применяемых в жизненном цикле и объединенных общим философским подходом.

С каждой методологией можно связать некоторые характерные для нее атрибуты:

- **Философский подход** (или основной принцип), определяющий основной источник эффективности методологии,
- Согласованное, связное **множество моделей методов**, через которые реализуется данная методология,
- **Концепции** (понятия, замыслы), поддерживающие методы и позволяющие более точно их определить.

Различные методологии программирования дают разный выигрыш для решения задач с различными характеристиками.

Некоторые широко распространенные задачи:

- Научно-технические расчеты (аналитические и численные модели),
- Экономические расчеты (доступ к базам данных и генерация отчетов),
- Системы реального времени, требующие контроля планирования выполнения,
- Обработка текстов, операции над строками символов,
- Графические пользовательские интерфейсы.

Этот выигрыш можно оценивать, в первую очередь, по общим затратам на разработку программного обеспечения. Следующий немаловажный фактор – эффективность работы созданного программного обеспечения на современных компьютерах.

Обязательно следует учесть, что некоторые методологии могут быть просто противопоказаны определенным задачам.

Профессиональное программирование

Методология программирования

Суть (ядро) методологий определяются способом описания алгоритмов.

Основные ядра методологии :

- **Методология императивного программирования**
- **Методология объектно-ориентированного программирования**
- **Методология функционального программирования**
- **Методология логического программирования**

Каждое из ядер может получить "приставку", определяемую некоторой топологией, которая может быть "хорошей" или "плохой". Топологическая специфика (топология) методологий – это способ выбора методов для получения уточненного ядра методологии.

Критерием качества топологий может быть количество общих затрат на разработку программного обеспечения. Затраты определяются совокупностью многочисленных факторов, в том числе связанных с абстракциями данных, управления и модульности. Так к "хорошей" топологии приводит отказ от использования глобальных данных и оператора безусловного перехода (за исключением особого ряда случаев), сильная связность модулей и их слабое сцепление.

Пример. Если в императивной методологии придерживаться методов структурного программирования (дающих хорошую топологию с позиций всех упомянутых абстракций), то получаем хорошо известную методологию структурного императивного программирования (краткое имя – методология структурного программирования). Одной из главных причин перехода к структурному программированию в начале 1970-х была необходимость снижения затрат на тестирование. Успех объектно-ориентированной методологии изначально определила ее хорошая топология, базирующаяся на абстрактных типах данных.

Каждое из ядер может получить "суффикс", уточняющий реализационную специфику методологий, которая определяется организацию аппаратной поддержки данной методологии. Наиболее известными организациями : централизованная и параллельная.

Примеры параллельных методологий:

- **Методология императивного параллельного программирования. Эту методологию обычно называют кратко – методология параллельного программирования.**
- **Методология логического параллельного программирования.**

Профессиональное программирование

Методология программирования

Методология императивного программирования

Методология императивного (процедурного) программирования – подход, характеризующийся принципом последовательного изменения состояния вычислителя пошаговым образом. При этом управление изменениями полностью определено и полностью контролируемо.

Императивное программирование – это исторически первая поддерживаемая аппаратно методология программирования. Она ориентирована на классическую фон Неймановскую модель, остававшуюся долгое время единственной аппаратной архитектурой, получившей широкое практическое применение.

Методы и концепции

- **Метод изменения состояний заключается в последовательном изменении состояний. Метод поддерживается концепцией алгоритма.**
- **Метод управления потоком исполнения заключается в пошаговом контроле управления. Метод поддерживается концепцией потока исполнения.**

Синтаксис и семантика

Основным синтаксическим понятием является оператор.

Первая группа – атомарные операторы, у которых никакая их часть не является самостоятельным оператором (например, оператор присваивания, оператор безусловного перехода, вызова процедуры и т. п.).

Вторая группа – структурные операторы, в которых другие операторы играют роль более крупных операторов (например, составной оператор, операторы выбора, цикла и т. п.).

Императивное программирование – это парадигма программирования, которая, описывает процесс вычисления в виде инструкций, изменяющих состояние данных. Императивная программа очень похожа на приказы, выражаемые повелительным наклонением в естественных языках, то есть это последовательность команд, которые должен выполнить компьютер.

Императивное программирование основано на изменении состояний вычислителя. Если под вычислителем понимать современный компьютер, то его состоянием будут значения всех ячеек памяти, состояние процессора (в том числе указатель текущей команды) и всех сопряженных устройств. Единственная структура данных – последовательность ячеек (пар "адрес -> значение") с линейно упорядоченными адресами.

Профессиональное программирование

Методология программирования

Методология императивного программирования

В качестве математической модели императивное программирование использует машину Тьюринга – абстрактное вычислительное устройство, предложенное на заре компьютерной эры для описания алгоритмов.

Краткое описание семантики императивного программирования выглядит так:

- ✔ **Операторы исполняются в порядке, предписанном объемлющем их структурным оператором.**
- ✔ **Если это составной оператор, то входящие в него операторы исполняются в том порядке, в котором записаны.**
- ✔ **Операторы, входящие в оператор выбора, могут исполняться или не исполняться в зависимости от значения логического условия.**
- ✔ **Исполнение атомарных операторов сводится к соответствующему единичному изменению состояния вычислителя.**
- ✔ **Исполнение вызова процедуры допускает возникновение по его ходу других вызовов.**
- ✔ **Традиционное средство структурирования – подпрограмма (процедура или функция). Подпрограммы имеют параметры и локальные определения и могут быть вызваны рекурсивно. Функции возвращают значения как результат своей работы.**

Профессиональное программирование

Методология программирования

Методология императивного программирования

Языки императивного программирования

Языки императивного программирования манипулируют данными в пошаговом режиме, используя последовательные инструкции и применяя их к разнообразным данным. Считается, что первым алгоритмическим языком программирования был язык Plankalkuel (от plan calculus), разработанный в 1945-1946 годах Конрадом Цузе (Konrad Zuse).

Наиболее известные и распространенные императивные языки программирования, большинство из которых было создано в конце 50-х – середине 70-х годов XX века, представлены в таблице. Пустое место на рисунке, соответствующее 1980 – 2000 годам прошлого века – это период увлечения новыми парадигмами, и чистых императивных языков в это время практически не появлялось.

ГОДЫ	Языки программирования
1950	Fortran (1954)
1960	Algol (1960)
1970	Pascal (1970); C (1972); ICON (1974)
1980 - 2000	

Класс задач

Императивное программирование наиболее пригодно для решения задач, в которых последовательное исполнение каких-либо команд является естественным. Примером здесь может служить управление современными аппаратными средствами. Поскольку практически все современные компьютеры императивны, эта методология позволяет порождать достаточно эффективный исполняемый код.

С ростом сложности задачи императивные программы становятся все менее и менее читаемыми. Программирование и отладка действительно больших программ (например, компиляторов), написанных исключительно на основе методологии императивного программирования, может затянуться на долгие годы.

Профессиональное программирование

Методология программирования

Методология объектно-ориентированного программирования

Методология объектно-ориентированного программирования – это системный подход, использующий объектную декомпозицию, при которой статическая структура системы описывается в терминах объектов и связей между ними, а поведение системы описывается в терминах обмена сообщениями между объектами.

Исследования в области моделирования реальных систем привели к необходимости создания средств естественного описания сущностей, которые в них встречаются: объектов и событий. Концепции инкапсуляция (абстрактные типы данных), наследование и полиморфизм являются полезным дополнением к традиционному структурному программированию.

Методы и концепции

- **Метод объектно-ориентированной декомпозиции заключается в выделении объектов и связей между ними. Метод поддерживается концепциями инкапсуляции, наследования и полиморфизма.**
- **Метод абстрактных типов данных – это метод, лежащий в основе инкапсуляции. Метод поддерживается концепцией абстрагирования.**
- **Метод пересылки сообщений заключается в описании поведения системы в терминах обмена сообщениями между объектами. Метод поддерживается концепцией сообщения.**

Синтаксис и семантика

Вычислительная модель чистого объектно-ориентированного программирования поддерживает явно только одну операцию, которой является посылка объекту сообщения. Сообщения могут иметь параметры, являющиеся объектами. Само сообщение также является объектом. Объект имеет набор обработчиков сообщений (набор методов). У объекта есть поля – персональные переменные для данного объекта, значениями которых являются ссылки на другие объекты. В одном из полей объекта хранится ссылка на объект-предок, которому переадресуются все сообщения, не обрабатываемые данным объектом. Структуры, описывающие обработку и переадресацию сообщений, обычно выделяют в отдельный объект, называемый классом данного объекта. Сам объект называют экземпляром указанного класса.

Профессиональное программирование

Методология программирования

Методология объектно-ориентированного программирования

Естественным средством структурирования в данной методологии являются классы. Классы определяют, какие поля и методы экземпляра доступны извне, как обрабатывать отдельные сообщения и т. п. В чистых объектно-ориентированных языках извне доступны только методы.

Взаимодействие задач в данной методологии осуществляется при помощи обмена сообщениями между объектами, реализующими данные задачи. Под объектом в ней подразумевается активный процесс, получающий сообщения и изменяющий свое внутреннее состояние в зависимости от полученного сообщения.

Для этой модели справедливы следующие свойства:

- объектом является процесс, который может иметь различные внутренние состояния. При получении сообщения объект становится активным;
- извне внутреннее состояние объекта может быть изменено только посредством передачи ему сообщения, специфицирующего выполняемую объектом операцию;
- во время работы объект может обмениваться сообщениями с другими объектами.

Профессиональное программирование

Методология программирования

Методология объектно-ориентированного программирования

Языки объектно-ориентированного программирования

Они содержат конструкции, позволяющие определять объекты, принадлежащие классам и обладающие свойствами инкапсуляции, наследования и полиморфизма.

Такие языки можно разделить на три группы (см. таблицу).

- **Чистые языки, в наиболее классическом виде поддерживающие объектно-ориентированную методологию. Такие языки обычно содержат существенную библиотеку, а также набор средств поддержки времени исполнения;**
- **Гибридные языки, которые появились в результате внедрения объектно-ориентированных конструкций в популярный императивный язык программирования;**
- **Урезанные (очищенные) языки, которые появились в результате удаления из гибридных языков некоторых конструкций (в т.ч, наиболее опасных и ненужных с объектно-ориентированной точки зрения).**

ГОДЫ	Чистые	Гибридные	Урезанные
1960	Simula (1962)		
1970	Smalltalk (1972); Beta (1975)		
1980	Self (1986)	C++ (1983); Object Pascal (1984)	
1990	Cecil (1992)		Java (1995)
2000			C# (2000)

Класс задач

Данная методология является мощным средством для моделирования отношений между объектами практически в любой предметной области. Особенно удобно и легко в объектах выразить взаимодействие между различными элементами графического интерфейса пользователя.

Категорически не рекомендуется применять эту методологию для систем реального времени с жесткими ограничениями на время.

Профессиональное программирование

Методология программирования

Методология функционального программирования

Методология функционального программирования – способ составления программ, в которых единственным действием является вызов функции, единственным способом расчленения программы на части – введение имени для функции и задание для этого имени выражения, вычисляющего значения функции, а единственным правилом композиции – оператор суперпозиции функции. Функциональное программирование часто называют аппликативным программированием поскольку основной его механизм - это аппликация (применение) функции к аргументам.

Функциональная методология является одной из старейших. По происхождению она тесно связана с лямбда-исчислением, изобретенным еще в начале 30-х годов XX века логиком Алонзо Черчем (Alonzo Church). Для многих функциональная методология стала ассоциироваться с языком Lisp, созданным Джоном Маккарти (John McCarthy) в конце 50-х годов XX века [Хюве-нен, Сеппянен 1986]. Эта методология в основном используется в программах искусственного интеллекта.

Методы и концепции

- **Метод аппликативности заключается в том, что программа есть выражение, составленное из применения функций к аргументам. Программа состоит из совокупности определений функций, представляющих собой вызовы других функций и вложенных друг в друга. Метод поддерживается концепцией функции.**
- **Метод рекурсивного поведения заключается в самоповторяющемся поведении, возвращающемся к самому себе. Метод поддерживается концепцией рекурсии.**
- **Метод настраиваемости заключается в том, что можно легко порождать новые программные объекты по образцу, как значения соответствующих выражений. Этому способствует то, что не только программа, но и любой программный объект (в идеале) является выражением.**

Профессиональное программирование

Методология программирования

Методология функционального программирования

Синтаксис и семантика

Функциональное (аппликативное) программирование радикально отличается от императивного (процедурного). **Функциональная программа** представляет собой **просто выражение**, а **выполнение программы – процесс его вычисления**. В функциональном программировании отсутствует понятие времени. При описании функционального программирования, как правило, рассматривают так называемое "**расширенное лямбда-исчисление**".

Пояснение: императивная программа детерминирована, т.е. выход полностью определяется входом, т.е. конечное состояние или тот его фрагмент, который нас интересует, являются функцией начального состояния, например a'=f(a) [в соответствии с замечанием Наура любую программу можно записать в виде одного выражения Output = Program(Input)].

Переменных в функциональном программировании – нет. Константами в расширенном лямбда-исчислении могут быть числа, кортежи, списки, имена предопределенных функций и т. п. Результатом вычисления применения предопределенной функции к аргументам будет значение предопределенной функции в этой "точке". Результатом применения лямбда-абстракции к аргументу будет подстановка аргумента в выражение – "тело" лямбда-абстракции. Сами лямбда-абстракции также являются выражениями и, следовательно, могут быть аргументами.

К **недостаткам ФП** можно отнести то, что функциональные программы не используют переменные - то есть не имеют состояния. Соответственно, они не могут использовать присваивание, поскольку нечему присваивать. Кроме того, идея последовательного выполнения операторов также бессмысленна, поскольку первый оператор не имеет никакого влияния на второй, так как нет никакого состояния, передаваемого между ними.

К **достоинствам функционального подхода** можно отнести то, что функциональные программы могут использовать функции более изящным способом. Функции могут рассматриваться точно так же, как и более простые объекты, такие как целые числа: они могут передаваться в другие функции как аргументы и возвращаться в качестве результатов, а также применяться в вычислениях. Вместо последовательного выполнения операторов и использования циклов, функциональные языки программирования предлагают рекурсивные функции, т.е. функции, определённые в терминах самих себя.

Профессиональное программирование

Методология программирования

Методология функционального программирования

Языки функционального программирования

Основная специфика языков функционального программирования заключается в том, что функции обмениваются между собой данными непосредственно, т. е. без использования промежуточных переменных и присваиваний. Переменные, однажды получив значение, никогда его не изменят.

В таблице представлены основные функциональные языки. Первый пик интереса к ним приходится на конец 70-х – начало 80-х годов XX века. Второй пик приходится на настоящее время, и он связан в первую очередь со стандартизацией наиболее мощного чистого функционального языка – Haskell.

ГОДЫ	Языки программирования
1960	Lisp (1958)
1970	РЕФАЛ (1968); Scheme (1975); FP(1977); ML (1978)
1980	Miranda (1985); Standard ML (1985)
1990	Haskell (1990,1998)

Класс задач

Функциональное программирование обычно применяется для решения тех задач, которые трудно сформулировать в терминах последовательных операций. В эту категорию попадают практически все задачи, связанные с искусственным интеллектом. Это такие задачи, как обработка естественного языка, экспертные консультирующие системы, проблемы зрительного восприятия и многие другие.

Профессиональное программирование

Методология программирования

Методология логического программирования

Методология логического программирования – подход, согласно которому программа содержит описание проблемы в терминах фактов и логических формул, а решение проблемы система выполняет с помощью механизмов логического вывода.

Логическое программирование начинает свой отсчет времени с конца 60-х годов XX века, когда Корделл Грин (Cordell Green) предложил использовать резолюцию как основу логического программирования. Алан Колмероэ (Alain Colmerauer) создал язык логического программирования Prolog в 1971 году. В основе логических языков лежит теория хорновских дизъюнктов. Логическое программирование пережило пик популярности в середине 80-х годов XX века, когда оно было положено в основу проекта разработки программного и аппаратного обеспечения вычислительных систем пятого поколения.

Методы и концепции

- **Метод единообразия** заключается в **одинаковом применении механизма логического доказательства ко всей программе.**
- **Метод унификации** – это механизм **сопоставления с образцом для создания и декомпозиции структур данных.**

Синтаксис и семантика

В логике теории задаются при помощи аксиом и правил вывода. То же самое мы имеем и в базисном языке логического программирования Prolog. Аксиомы здесь принято называть фактами, а правила вывода ограничить по форме до так называемых "дизъюнктов Хорна" – утверждений вида a <= B1&…&Bn. В языке Prolog такие утверждения принято записывать следующим образом: a :-b1, ..., bn.Факты (они же аксиомы) представляются как правила с пустой "посылкой": a. Переменные в утверждениях принято обозначать идентификаторами, начинающимися с заглавной буквы. Утверждение, которое требуется доказать в Prolog принято называть целями. Prolog-система использует метод унификации и метод резолюций для доказательства утверждений.

Унификация – это сопоставление двух произвольных термов, содержащих переменные, с целью определения того, можно ли присвоить этим переменным такие значения, чтобы получились два одинаковых терма. Например, унификация термов f(X, 2) и f(1, Y), где X, Y – переменные, выдаст подстановку: X=1, Y=2. Унификация термов f (X) и X пройдет безуспешно.

Метод резолюций заключается в последовательном доказательстве отдельных утверждений, входящих в посылку дизъюнкта Хорна, для доказательства его следствия. То есть применение метода резолюций к правилу a:-b,c и утверждению a приведет к последовательному доказательству утверждений b и c. Метод резолюций имеет прямой аналог в обычной логике высказываний – правило modus ponens, по которому (a & (a=>b)) => b.

Профессиональное программирование

Методология программирования

Методология логического программирования

Языки логического программирования

Они содержат конструкции, позволяющие выполнить описание проблемы в терминах фактов и логических формул, а собственно решение проблемы выполняет система с помощью механизмов логического вывода. От языка программирования неотъемлем механизм конструктивного вывода целевого утверждения, основанный на строгих математических моделях. Язык Prolog является родоначальником семейства ему подобных языков (см. таблицу), в котором можно выделить три ветви.

- **Модификации языка (использование более мощных логических средств, внесение модульности и т. п.).**
- **Функциональное направление (комбинация с функциональными языками).**
- **Параллельное направление (логическое программирование по сути своей параллельно).**

ГОДЫ	Модификации	Функциональные	Параллельные
1970	Prolog (1971)		
1980	Prolog II (1980) IC-Prolog (1982) Mprolog (1983) T-Prolog (1983) LQF (1984)	L0GLISP(1982) LCA (1982) LEAF (1985)	Concurrent Prolog (1983) PARLOG (1983) GHC (1986)
1990	Goedel (1992)	Mercury (1993)	

Класс задач

Класс задач логического программирования практически совпадает с классом задач функционального программирования.

Профессиональное программирование

Методология программирования

Топологическая специфика методологий. Методология структурного императивного программирования

Методология структурного императивного программирования – подход, заключающийся в задании хорошей топологии императивных программ, ориентированной на сокращение количества общих затрат на разработку программного обеспечения.

Сокращение будет иметь место в результате того, что и проектные модели, и программный код будут иметь хорошую структурированность, позволяющую избежать многих ошибок. В случае данной методологии хорошую топологию задают отказ от использования глобальных данных и, в большинстве случаев, оператора безусловного перехода, разработка модулей с сильной связностью и обеспечение их независимости от других модулей.

Подход базируется на двух основных принципах построения:

- последовательная декомпозиция алгоритма решения задачи сверху вниз;
- использование структурного кодирования.

Создателем структурного подхода считается Эдсгер Дейкстра.

Методы и концепции

- Метод алгоритмической декомпозиции сверху вниз заключается в пошаговой детализации постановки задачи, начиная с наиболее общей задачи. Данный метод обеспечивает хорошую структурированность и поддерживается концепцией алгоритма.
- Метод модульной организации частей программы заключается в разбиении программы на специальные компоненты, называемые модулями. Метод поддерживается концепцией модуля.
- Метод структурного кодирования заключается в использовании при кодировании трех основных управляющих конструкций. Метки и оператор безусловного перехода являются трудно отслеживаемыми связями, без которых мы хотим обойтись. Метод поддерживается концепцией управления.

Языки структурного программирования

Основное отличие от классической методологии императивного программирования заключается в структурности программ и в трех способах композиции операторов: последовательности двух или более операторов, дихотомического и множественного выбора, а также повторения.

Класс задач

Класс задач для данной методологии соответствует классу задач для императивной методологии. Заметим, что при этом удается разрабатывать более сложные программы, поскольку их легче воспринимать и анализировать.

Профессиональное программирование

Методология программирования

Реализационная специфика методологий

Методология императивного параллельного программирования

Методология императивного параллельного программирования – подход, в котором предлагается использование явных конструкций для параллельного исполнения выбранных фрагментов программ.

Вычислительные задачи часто имеют огромные объемы. Анализ эффективности их решений показал, что ситуацию можно значительно улучшить, если использовать для вычислений не одно, а несколько вычислительных устройств одновременно. Создание аппаратных многопроцессорных архитектур привело к широким исследовательским работам в этой области. Еще одна причина возникновения данной методологии связана с появлением достаточно сложных программ, требующих поддержки явного параллелизма (например, операционных систем).

Методы и концепции

Метод синхронизации исполняемого кода заключается в использовании специальных атомических операций для осуществления взаимодействия между одновременно исполняемыми фрагментами кода. Метод поддерживается концепцией примитивов синхронизации.

Методология логического параллельного программирования

Логическое программирование допускает естественную параллельную реализацию. В примере a:- b,c. порядок согласования целей b и c не имеет значения, поэтому их можно доказывать параллельно.

Процессы доказательства b и c образуют И-систему процессов. И-система успешно доказывается, если каждый процесс, входящий в систему, успешен. В примере с предикатом member два правила для него могли применяться параллельно, образуя ИЛИ-систему процессов. ИЛИ-система успешно доказывается, если хотя бы один процесс в системе успешен. Переменные, общие для системы процессов (например, в случае a(x) :- b(x), c(x) .), преобразуются в каналы связи между процессами в системе. Связывание переменной (присвоение ей значения) аналогично посылке значения в канал.

Профессиональное программирование

Технологии программирования

Классические технологические процессы
- Анализ требований и проектирование
- Программирование (реализация)
- Тестирование и отладка
- Ввод программы в действие
- Эксплуатация и сопровождение
- Завершение эксплуатации

Стандартные технологические процессы
- Основные процессы
- Вспомогательные процессы
- Организационные процессы
- Взаимосвязь между процессами

Основные стадии технологических подходов
- Фазы как крупные временные рамки
- Стадии как отражение классических процессов
- Контрольные точки

Основные технологические подходы
- Каскадные технологические подходы
- Каркасные технологические подходы
- Генетические технологические подходы
- Подходы на основе формальных преобразований
- Группа ранних подходов быстрой разработки
- Адаптивные технологические подходы

Технологии коллективной разработки
- Авторская разработка
- Коллективная разработка
- Общинная модель разработки
- Отступление "об оффшорном программировании"

Качество программного обеспечения
- Характеристики качества программного обеспечения
- Оценка качества процесса разработки
- "Достаточно хорошее" программное обеспечение
- Стандартизация информационных технологий

Языки программирования

Языковые абстракции
Отступление "об абстрагировании"
Абстракция данных
Абстракция управления
Абстракция модульности

Языки моделирования
Моделирование на основе структурной методологии
Моделирование на основе объектно-ориентированной методологии
Языки моделирования данных
Языки моделирования знаний

Языки программирования высокого уровня
Обзор языков, принадлежащих к семействам
Обзор языков, ориентированных на предметную область

Формальные языки
Иерархия грамматик
Техники распознавания (разбора)
Образное сравнение типов грамматик
Метатрансляция
Язык спецификации грамматик

Естественные языки
Особенности естественных языков и культурных сред
Семантический анализ естественных языков
Интернационализация и локализация программных продуктов

Профессиональное программирование

Системы программирования

- Процесс-ориентированный инструментарий
- Универсальный инструментарий
  - Инструменты работы с текстом
  - Системы документирования
  - Системы разработки интерфейсов
  - Системы управления базами данных
  - Системы управления базами знаний и экспертные системы
  - Электронные библиотеки и инструментарий Интернета
- Инструментарий поддержки процессов некоторых технологических подходов
  - Системы верификации программ
  - Системы формального преобразования программ
  - Средства сборочного программирования
- Инструментальные системы
  - Инструментальные среды программирования
  - Средства автоматизации разработки программ (CASE-средства)
  - Интегрированные среды
  - Репозитории проекта
- Средства поддержки коллективной разработки
  - Системы разделения файлов
  - Системы поддержки работы виртуальных групп
- Естественно-языковый интерфейс
  - Диалоговые системы
  - Вопросно-ответные системы
  - Автоматизированные обучающие системы и системы контроля знаний
  - Системы искусственного интеллекта

Профессиональное программирование

Архитектурная платформа

Основы архитектуры ЭВМ
Основные архитектуры набора команд
Классические архитектуры
Архитектуры CISC, RISC и VLIW
Вычисления
Архитектура ЭВМ и языки программирования
Оценка производительности вычислительных систем
Особенности SPARC-архитектуры
Основные понятия SPARC-архитектуры
Основной набор команд языка Assembler
Примеры программ на языке Assembler
Организация SPARC-архитектуры
Параллельные и распределенные архитектуры
Основные классы параллельных архитектур
Коммутаторы вычислительных систем
Некоторые классификации параллельных архитектур
Архитектура компьютерных сетей
Классификация сетей и сетевые топологии
Стандарты в области сетей IEEE 802
Аппаратная поддержка локальных сетей
Глобальная сеть Интернет

Профессиональное программирование

Операционная платформа

- История и эволюция операционных систем
- Классификация операционных систем
- Процессы
  - Процессы и потоки (нити) управления
  - Коммуникация и синхронизация процессов в централизованных архитектурах
  - Коммуникация процессов в сетях
  - Синхронизация процессов в распределенных системах
- Память
  - Основная память
  - Внешняя память
- Драйверы устройств