Технология программирования (ТП)

ТП - совокупность методов и средств, используемых в процессе разработки ПО.

ТП представляет собой набор технологических инструкций, включающих:

- указание последовательности выполнения технологических операций;
- перечисление условий, при которых выполняется та или иная операция;
- описания самих операций, для каждой операции определены исходные данные, результаты, а также инструкции, нормативы, стандарты, критерии и методы оценки и т.п.

Технология также определяет способ описания проектируемой системы, точнее модели, используемой на конкретном этапе разработки.

Основные этапы программирования как науки

1 этап - «Стихийное» программирование

(до середины 60х годов XX века)

2 этап - Структурный подход к программированию (60-70-е годы XX в.)

В результате существенно увеличивается

3 этап - Объектный подход к программированию

(с середины 80-х до конца 90-х годов)

4 этап - Компонентный подход и CASE-технологии (с середины 90-х годов XX в. до наст. времени)

«Стихийное» программирование

- отсутствие технологий, программирование - искусство.

Программы простейшей структуры: состояли из программы на машинном языке и обрабатываемых ею данных:

.

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

«Стихийное» программирование

Появление ассемблеров - вместо двоичных и 16-ричных кодов стали использоваться символические имена данных и мнемоники кодов операций → более «читаемые» программы

Создание ЯП высокого уровня (FORTRAN, ALGOL) упростило программирование вычислений, снизив уровень детализации операций.

Это позволило увеличить сложность программ

«Стихийное» программирование

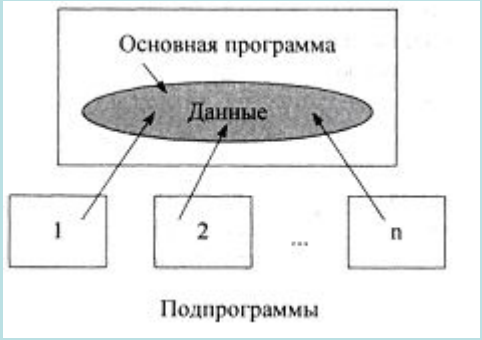
Революционным стало появление средств работы с подпрограммами (п/п).

(Идея написания п/п появилась раньше, но отсутствовали средства поддержки) снижало эффективность их применения.)

П/п можно было сохранять и использовать в других программах. В результате были созданы огромные библиотеки расчетных и служебных подпрограмм.

«Стихийное» программирование

Программа состояла из основной программы, области *глобальных данных* и набора п/п (библиотечных), выполняющих обработку всех данных или их части.



Слабое место - при ↑числа п/п возрастала вероятность искажения части глобальных данных какой-либо подпрограммой

«Стихийное» программирование

Для ↓количество таких ошибок - в п/п размещать ***локальные данные:***

Сложность разрабатываемого ПО по-прежнему ограничивалась возможностью программиста отслеживать процессы обработки данных, но уже на новом уровне.

Но появление средств поддержки п/п позволило осуществлять разработку ПО нескольким программистам параллельно.

«Стихийное» программирование

В начале 60-х г XX в.

- «кризис программирования»:

фирмы, разрабатывающие сложное ПО, срывали сроки завершения проектов.

Проект устаревал раньше, чем был готов к внедрению, увеличивалась его стоимость, многие проекты никогда не были завершены.

«Стихийное» программирование

Объективная причина - несовершенство ТП.

Стихийно использовалась разработка «снизу-вверх»: вначале проектировали и реализовывали простые п/п, из них строили сложную программу.

Без четких моделей описания и методов проектирования создание п/п – сложная задача, интерфейсы получались сложными, при сборке программы - ошибки согласования.

Исправление ошибок ⟶ изменение п/п ⟶ в п/п вносились новые ошибки, которые необходимо исправлять...

В итоге процесс тестирования и отладки > 80% времени разработки или вообще не заканчивался.

Встал вопрос ***разработки технологии создания сложных программных продуктов***, снижающей вероятность ошибок проектирования.

Структурный подход

- совокупность рекомендуемых технологических приемов, охватывающих выполнение всех этапов разработки ПО.

В основе - ***декомпозиция*** сложных систем с целью последующей реализации в виде отдельных небольших (до 50 операторов) п/п.

С появлением других принципов декомпозиции (объектной, логической и т.д.) данный способ получил название ***процедурной*** декомпозиции.

Структурный подход

Структурный подход - представление задачи в виде иерархии подзадач простейшей структуры. Проектирование - ***«сверху-вниз»:*** реализация общей идеи + проработка интерфейсов п/п.

Вводились ограничения на конструкции алгоритмов, рекомендовались формальные модели их описания, метод проектирования алгоритмов - ***пошаговой детализации.***

Поддержка принципов структурного программирования была заложена в основу *процедурных ЯП*.

Они включали основные «структурные» операторы передачи управления, поддерживали вложение подпрограмм, локализацию и ограничение области «видимости» данных.

ЯП: PL/1, ALGOL-68, Pascal, C.

Структурный подход

Одновременно появилось множество языков, базирующихся на других концепциях. Большинство из них не выдержало конкуренции, какие-то были забыты, идеи других были в использованы в следующих версиях развиваемых языков.

Дальнейший рост сложности и размеров ПО потребовал развития ***структурирования данных***.

Появляется возможность определения ***пользовательских типов данных***.

Для ↓количество ошибок - стремление разграничить доступ к глобальным данным программы.

Появилась и начала развиваться технология ***модульного программирования***.

Модульное программирование

- выделение групп п/п с общими глобальными данными в отдельно компилируемые *модули* (библиотеки п/п): модуль графических ресурсов и др.

Связи между модулями - через специальный интерфейс, доступ к реализации модуля запрещен. Эту технологию поддерживают современные версии языков Pascal, C++, языки Ада и Modula

Модульное программирование

Модульное программирование упростило разработку ПО несколькими программистами. Каждый мог разрабатывать свои модули независимо, обеспечивая взаимодействие модулей через межмодульные интерфейсы. Модули можно было использовать в других разработках, это повысило производительность труда программистов.

Практика показала, что структурный подход в сочетании с модульным программированием позволяет получать надежные программы размером *до 100.000 операторов*.

Узкое место - ошибка в интерфейсе выявляется только при выполнении п/п (из-за раздельной компиляции модулей). При ↑ размера программы возрастает сложность межмодульных интерфейсов

Для разработки ПО большого объема было предложено использовать ***объектный подход***.

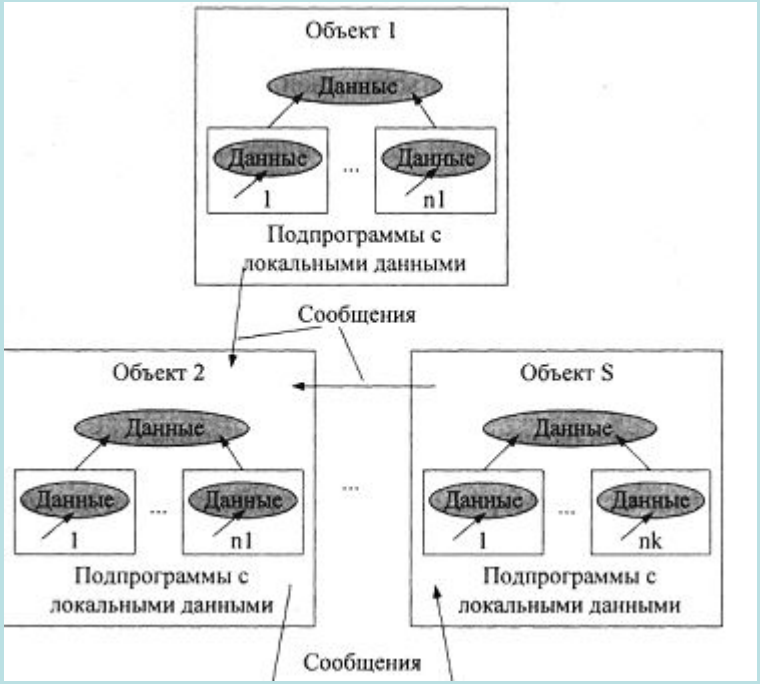
Объектный подход:

Программа - совокупность О.

Каждый О - экземпляр класса.

Классы - иерархию с наследованием свойств.

Взаимодействие О - путем передачи сообщений.



Объектная структура программы - впервые в языках имитационного моделирования Simula (60-е г.), Smalltalk (70-е г.), затем - в новых версиях ЯП: Pascal, C++, Modula, Java.

Объектный подход

Достоинство ООП *- «более естественная» декомпозиция ПО , это облегчает разработку → более полная локализация данных → возможность вести независимую разработку отдельных частей (объектов) программы.*

Объектный подход - новые способы организации программ на механизмах наследования, полиморфизма, инкапсуляции, композиции, что позволяет конструировать сложные объекты из простых.

В результате - ↑ показатель повторного использования кодов, возможность создания библиотек классов.

Объектный подход

Развитие ООП: созданы визуальные среды: Delphi, C++ Builder, Visual C++ для проектирования интерфейсов. Результат - заготовка программы, в которую уже внесены коды.

Недостатки ООП в конкретных реализациях ЯП:

- нет стандартов компоновки результатов компиляции объектов в единое целое → необходимость разработки ПО с использованием одного языка ЯП и компилятора → необходимо наличие исходных кодов библиотек классов;
- изменение реализации одного программного О связано с перекомпиляцией модуля и перекомпоновкой всего ПО.

Сохраняется зависимость модулей от адресов, полей, методов, структур и форматов данных. Модули взаимодействуют между собой. Связи модулей нельзя разорвать, но можно стандартизировать их взаимодействие - на этом основан ***компонентный подход к программированию.***

Компонентный подход

- построение ПО из компонентов – физически отдельных частей ПО, взаимодействие - через стандартизованные двоичные интерфейсы.

Объекты-компоненты можно собрать в динамически вызываемые библиотеки (DLL), исполняемые файлы, распространять в двоичном виде без исходных текстов, использовать в ЯП, поддерживающем технологию.

Рынок объектов - реальность, в Internet – множество компонентов → возможность создания программ, состоящих из повторно использованных частей.

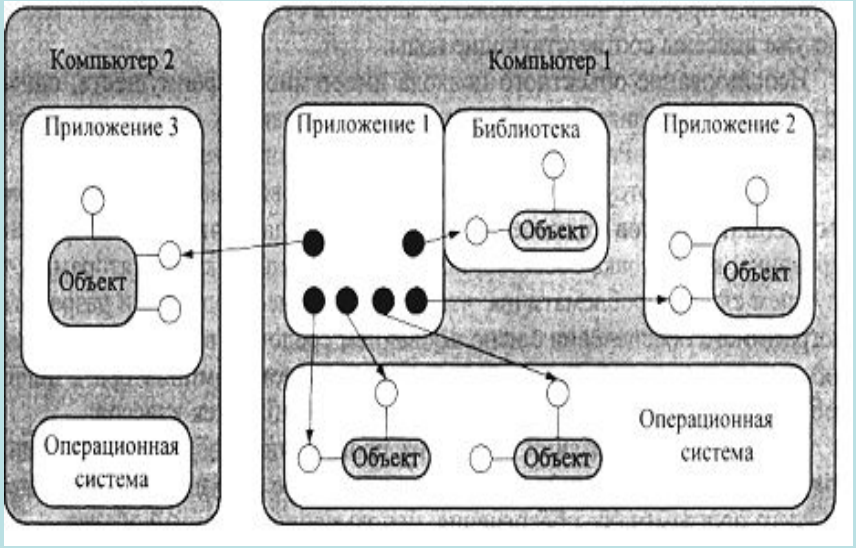
Компонентный подход - в основе **COM-технологий** (*Component Object Model*), и технологии создания распределенных приложений **CORBA** (*Common Object Request Broker Architecture*).

Технологии используют сходные принципы и различаются лишь особенностями их реализации.

Компонентный подход

Технология ***COM*** (Microsoft) - развитие технологии OLEI (Object Linking and Embedding – связывание и внедрение объектов)

определяет *общую парадигму взаимодействия программ любых типов:* библиотек, приложений, ОС; позволяет одной части ПО использовать функции *(службы),* другой, независимо от того, где функционируют ли эти части: в пределах одного процесса, в разных процессах на одном компьютере или на разных компьютерах



DCOM (Distributed COM) – распределенная COM для передачи вызовов между ПК -

Компонентный подход

На базе COM и DCOM:

- ***OLE-automation*** - технология создания программируемых приложений для доступа к внутренним службам приложений

ActiveX – на базе OLE-automation для создания ПО на одном и распределенных в сети ПК. Использует компоненты – элементы управления ActiveX

Преимущества ActiveX:

- быстрое написание программного кода;
- открытость и мобильность – спецификации технологии в Open Group как основа открытого стандарта;
- написание приложений с использованием знакомых средств разработки;
- большое количество бесплатных программных элементов ActiveX;
- стандартность - ActiveX основана на стандартах Internet (TCP/IP, HTML, Java) и стандартах COM, OLE.

Компонентный подход

- ***MTS*** (Microsoft Transaction Server – сервер управления транзакциями) – технология безопасной и стабильной работы распределенных приложений при больших объемах передаваемых данных.
- ***MIDAS*** (Multitier Distributed Application Server - сервер многозвенных распределенных приложений) – технология, организующая доступ к данным разных ПК с учетом балансировки нагрузки сети.

Технология **CORBA** (разработка OMG - Object Management Group) - подход, аналогичный COM, но на базе объектов и интерфейсов CORBA. Программное ядро реализовано для всех основных аппаратных и программных платформ. ***CORBA - для создания распределенного ПО в гетерогенной вычислительной среде***.

CASE - технологии

Особенность современного этапа развития ТП - ***создание и внедрение автоматизированных технологий разработки и сопровождения программного обеспечения - CASE-технологий*** (Computer-Aided Software/System Engineering).

Без средств автоматизации разработка сложного ПО невозможна: человек не в состоянии фиксировать все детали, необходимые при разработке ПО.

Существуют CASE-технологии, поддерживающие структурный, объектный и компонентный подходы к программированию.

Проблемы разработки сложных программных систем

Современные программные системы объективно очень сложны. Главная причина - ***логическая сложность решаемых задач***

В процесс компьютеризации вовлекаются новые предметные области, а для освоенных усложняются постановки задач.

Другие факторы, увеличивающие сложность разработки ПС:

- сложность формального определения требований к ПС;
- отсутствие средств описания поведения дискретных систем с большим числом состояний при недетерминированной последовательности входных воздействий;
- коллективная разработка;
- необходимость увеличения степени повторяемости кодов.

Блочно-иерархический подход к созданию сложных систем

В сложных системах - иерархическая внутренняя структура: связи элементов различны по типу и по силе,

система - совокупность взаимозависимых подсистем.

Внутренние связи элементов подсистем сильнее связей между подсистемами.

Подсистемы разделяют на подсистемы и т.д. до нижнего «элементарного» уровня, где система состоит из немногих типов подсистем, по-разному организованных.

Иерархии такого типа - ***«целое-часть».***

Поведение системы сложнее поведения отдельных частей, особенности системы обусловлены *отношениями* между ее частями, а не частями как таковыми.

Блочно-иерархический подход к созданию сложных систем

.

Другой вид иерархии – ***«простое - сложное»:*** любая система - результат развития более простой системы.

Этот вид иерархии реализуется механизмом наследования ООП.

Программные системы также иерархические.

На этих свойствах - ***блочно-иерархический подход*** к исследованию и созданию систем: сначала создаются части объектов, затем из них собирается объект.

Процесс разбиения сложного объекта на независимые части - ***декомпозиция***.

В процессе декомпозиции необходимо определить все виды связей частей между собой.

Блочно-иерархический подход к созданию сложных систем

. При создании сложных объектов - многократная декомпозиция - метод ***пошаговой детализации.***

Выделяют аналогичные блоки, это ↑ степень повторяемости кодов и ↓ стоимость разработки.

Результат декомпозиции - схема ***иерархии,*** на нижнем уровне - простые блоки, на верхнем – сам объект.

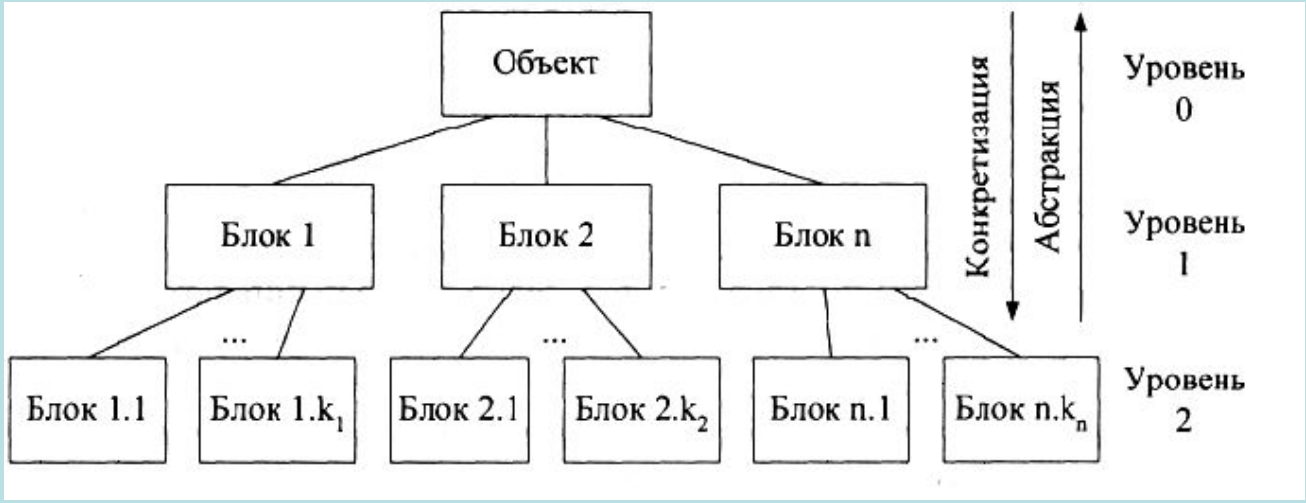
На каждом уровне - описание блоков с определенной степенью детализации, ***абстрагируясь*** от несущественных деталей.

Для каждого уровня - свои формы документации и свои модели.
Для объекта в целом - общие требования, блоки нижнего уровня специфицируют так, чтобы из них можно собрать работающий объект.

***Чем больше блок, тем более абстрактно его описание*.**

Блочно-иерархический подход к созданию сложных систем

.



Соотношение абстрактного и конкретного в описании блоков при блочно-иерархическом подходе

Блочно-иерархический подход к созданию сложных систем

В основе блочно-иерархического подхода - декомпозиция и иерархическое упорядочение.

Другие принципы:

- непротиворечивость - контроль согласованности элементов;
- полнота - контроль на присутствие лишних элементов;
- формализация – строгость методического подхода;
- повторяемость – выделение одинаковых блоков;
- локальная оптимизация – в пределах уровня иерархии.

Совокупность языков моделей, постановок задач, методов описаний иерархического уровня - *уровень проектирования.*

Блочно-иерархический подход:

- упрощает проверку работоспособности системы и блоков;
- обеспечивает возможность модернизации систем.