**Обзор современных основных инструментальных средств разработки программных продуктов.**

***ПО*** – совокупность программ и сопровождающей их документации, обеспечивающую решение прикладных задач и функционирование аппаратных средств ЭВМ.

К ПО относится область деятельности по его проектированию и разработке:

* технология проектирования программ;
* методы тестирования программ;
* анализ качества работы программ;
* документирование программ.

**Сфера применения ПО**

ПО современных компьютеров включает миллионы программ – от игровых до научных.

**ПО по назначению делится на:**

* Базовое (системное) ПО;
* Рабочее (прикладное) ПО;
* Инструментальное ПО (средства разработки ПО – СУБД реляционные (Oracle, MySQL), объектно-ориентированные, иерархические, сетевые).

**Классификация ПО по способу распространения**

* Коммерческое ПО;
* Бесплатные программы;
* Условно-бесплатные (их можно получить и опробовать бесплатно, но для систематического пользования нужно платить);
* Пиратские (ворованные) копии программ (не имеют документации).

**Пакеты прикладных программ**

* ППП – комплект программ, предназначенных для решения задач в определенной области
* Выделяет следующие виды ППП:
* проблемно-ориентированные (где возможна типизация функций управления) – ППП автоматизации бухучета, управления персоналом;
* Автоматизации проектирования (в работе конструкторов – разработка чертежей);
* Общего назначения (графические редакторы, СУБД);
* Офисные;
* Системы искусственного интеллекта (экспертные системы, поддержка общения на естественном языке).

**Программные средства и продукты**

* Программные средства – математические средства, с помощью которых решаются задачи автоматизированного получения, обработки, хранения и выдачи информации.
* Программный продукт – совокупность отдельных программных средств, их документации, гарантий качества, рекламных материалов, мер по обеспечению пользователей, распространению и сопровождению готового ПО.
* Программное изделие – программа или логически связанная совокупность программ, записанная на носителях данных, являющаяся продуктом промышленного производства, снабженная программной документацией, предназначенная для широкого распространения посредством продажи.

При этом путь от «программ для себя» до программных продуктов достаточно долгий. Программные продукты могут создаваться как индивидуальная разработка под заказ, разработка для массового распространения среди пользователей.

* При индивидуальной разработке фирма-разработчик создает оригинальный программный продукт, учитывающий специфику обработки данных для конкретного заказчика.
* При разработке для массового распространения фирма-разработчик, с одной стороны, должна обеспечить универсальность выполняемых функций обработки данных, с другой стороны, гибкость и настраиваемость программного продукта на условия конкретного применения.

Программный продукт создается на основе промышленной технологии выполнения проектных работ с применением современных инструментальных средств программирования. Специфика заключается в уникальности процесса разработки алгоритмов и программ. На создание программных продуктов затрачиваются значительные ресурсы – трудовые, материальные, финансовые, требуется высокая квалификация разработчиков.

Как правило, программные продукты требуют сопровождения, которое осуществляется специализированными фирмами – распространителями программ (дистрибьюторами). Сопровождение программ массового применения сопряжено с большими трудозатратами – обнаружение и исправление ошибок, создание новых версий программ и т.п.

**Программирование —** сравнительно молодая и быстро развивающаяся отрасль науки и техники. Опыт ведения реальных разработок и совершенствования имеющихся программных и технических средств постоянно переосмысливается, в результате чего появляются новые методы, методологии и технологии, которые, в свою очередь, служат основой более современных средств разработки программного обеспечения. Исследовать процессы создания новых технологий и определять их основные тенденции целесообразно, сопоставляя эти технологии с уровнем развития программирования и особенностями имеющихся в распоряжении программистов программных и аппаратных средств.

***Технологией программирования*** называют совокупность методов и средств, используемых в процессе разработки программного обеспечения. Как любая другая технология, технология программирования представляет собой набор технологических инструкций, включающих:

• указание последовательности выполнения технологических операций;

• перечисление условий, при которых выполняется та или иная операция;

• описания самих операций, где для каждой операции определены исходные данные, результаты, а также инструкции, нормативы, стандарты, критерии и методы оценки и т. п. (рисунок 1.1).

Кроме набора операций и их последовательности, технология также определяет способ описания проектируемой системы, точнее модели, используемой на конкретном этапе разработки.

Различают технологии, используемые на конкретных этапах разработки или для решения отдельных задач этих этапов, и технологии, охватывающие несколько этапов или весь процесс разработки. В основе первых, как правило, лежит ограниченно применимый метод, позволяющий решить конкретную задачу. В основе вторых обычно лежит базовый метод или подход, определяющий совокупность методов, используемых на разных этапах разработки, или методологию.

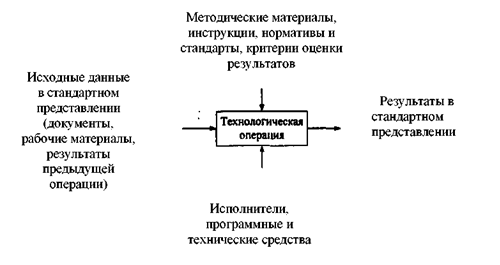


Рисунок 1.1 – Структура описания технологической операции

Чтобы разобраться в существующих технологиях программирования и определить основные тенденции их развития, целесообразно рассматривать эти технологии в историческом контексте, выделяя основные этапы развития программирования, как науки.

***Первый этап - «стихийное» программирование.*** Этот этап охватывает период от момента появления первых вычислительных машин до середины 60-х годов XX в. В этот период практически отсутствовали сформулированные технологии, и программирование фактически было искусством. Первые программы имели простейшую структуру. Они состояли из собственно программы на машинном языке и обрабатываемых ею данных (рисунок 1.2). Сложность программ в машинных кодах ограничивалась способностью программиста одновременно мысленно отслеживать последовательность выполняемых операций и местонахождение данных при программировании.

Появление ассемблеров позволило вместо двоичных или 16-ричных кодов использовать символические имена данных и мнемоники кодов операций. В результате программы стали более «читаемыми».

Создание языков программирования высокого уровня, таких, как FORTRAN и ALGOL, существенно упростило программирование вычислений, снизив уровень детализации операций. Это, в свою очередь, позволило увеличить сложность программ.

Революционным было появление в языках средств, позволяющих оперировать подпрограммами.

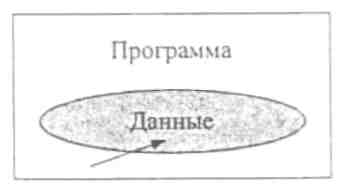


Рисунок 1.2 – Структура первых программ

(Идея написания подпрограмм появилась гораздо раньше, но отсутствие средств поддержки в первых языковых средствах существенно снижало эффективность их применения.) Подпрограммы можно было сохранять и использовать в других программах. В результате были созданы огромные библиотеки расчетных и служебных подпрограмм, которые по мере надобности вызывались из разрабатываемой программы.

Типичная программа того времени состояла из основной программы, области глобальных данных и набора подпрограмм (в основном библиотечных), выполняющих обработку всех данных или их части (рисунок 1.3).

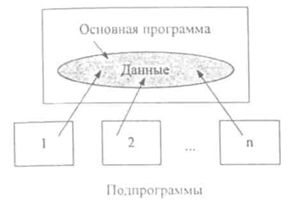


Рисунок 1.3 – Архитектура программы с глобальной областью данных

Слабым местом такой архитектуры было то, что при увеличении количества подпрограмм возрастала вероятность искажения части глобальных данных какой-либо подпрограммой. Например, подпрограмма поиска корней уравнения на заданном интервале по методу деления отрезка пополам меняет величину интервала. Если при выходе из подпрограммы не предусмотреть восстановления первоначального интервала, то в глобальной области окажется неверное значение интервала. Чтобы сократить количество таких ошибок, было предложено в подпрограммах размещать локальные данные (рисунок 1.4).

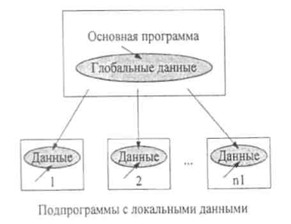


Рисунок 1.4 – Архитектура программы, использующей подпрограммы с локальными данными

Сложность разрабатываемого программного обеспечения при использовании подпрограмм с локальными данными по-прежнему ограничивалась возможностью программиста отслеживать процессы обработки данных, но уже на новом уровне. Однако появление средств поддержки подпрограмм позволило осуществлять разработку программного обеспечения нескольким программистам параллельно.

В начале 60-х годов XX в. разразился «кризис программирования». Он выражался в том, что фирмы, взявшиеся за разработку сложного программного обеспечения, такого, как операционные системы, срывали все сроки завершения проектов. Проект устаревал раньше, чем был готов к внедрению, увеличивалась его стоимость, и в результате многие проекты так никогда и не были завершены.

Объективно все это было вызвано несовершенством технологии программирования. Прежде всего стихийно использовалась разработка «снизу-вверх» - подход, при котором вначале проектировали и реализовывали сравнительно простые подпрограммы, из которых затем пытались построить сложную программу. В отсутствии четких моделей описания подпрограмм и методов их проектирования создание каждой подпрограммы превращалось в непростую задачу, интерфейсы подпрограмм получались сложными, и при сборке программного продукта выявлялось большое количество ошибок согласования. Исправление таких ошибок, как правило, требовало серьезного изменения уже разработанных подпрограмм, что еще более осложняло ситуацию, так как при этом в программу часто вносились новые ошибки, которые также необходимо было исправлять... В конечном итоге процесс тестирования и отладки программ занимал более 80 % времени разработки, если вообще когда-нибудь заканчивался. На повестке дня самым серьезным образом стоял вопрос разработки технологии создания сложных программных продуктов, снижающей вероятность ошибок проектирования.

Анализ причин возникновения большинства ошибок позволил сформулировать новый подход к программированию, который был назван «структурным».

***Второй этап - структурный подход к программированию*** (60-70-е годы XX в.).

Структурный подход к программированию представляет собой совокупность рекомендуемых технологических приемов, охватывающих выполнение всех этапов разработки программного обеспечения. В основе структурного подхода лежит декомпозиция (разбиение на части) сложных систем с целью последующей реализации в виде отдельных небольших (до 40 - 50 операторов) подпрограмм. С появлением других принципов декомпозиции (объектного, логического и т. д.) данный способ получил название процедурной декомпозиции.

В отличие от используемого ранее процедурного подхода к декомпозиции, структурный подход требовал представления задачи в виде иерархии подзадач простейшей структуры. Проектирование, таким образом, осуществлялось «сверху вниз» и подразумевало реализацию общей идеи, обеспечивая проработку интерфейсов подпрограмм. Одновременно вводились ограничения на конструкции алгоритмов, рекомендовались формальные модели их описания, а также специальный метод проектирования алгоритмов - метод пошаговой детализации.

Поддержка принципов структурного программирования была заложена в основу так называемых процедурных языков программирования. Как правило, они включали основные «структурные» операторы передачи управления, поддерживали вложение подпрограмм, локализацию и ограничение области «видимости» данных. Среди наиболее известных языков этой группы стоит назвать PL/1, ALGOL-68, Pascal, С.

Одновременно со структурным программированием появилось огромное количество языков, базирующихся на других концепциях, но большинство из них не выдержало конкуренции. Какие-то языки были просто забыты, идеи других были в дальнейшем использованы в следующих версиях развиваемых языков.

Дальнейший рост сложности и размеров разрабатываемого программного обеспечения потребовал развития структурирования данных. Как следствие этого в языках появляется возможность определения пользовательских типов данных. Одновременно усилилось стремление разграничить доступ к глобальным данным программы, чтобы уменьшить количество ошибок, возникающих при работе с глобальными данными. В результате появилась и начала развиваться технология модульного программирования.

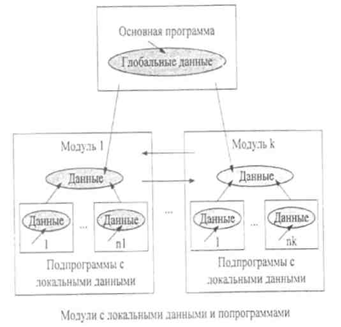


Рисунок 1.5 – Архитектура программы, состоящей из модулей

Модульное программирование предполагает выделение групп подпрограмм, использующих одни и те же глобальные данные в отдельно компилируемые модули (библиотеки подпрограмм), например, модуль графических ресурсов, модуль подпрограмм вывода на принтер (рисунок 1.5). Связи между модулями при использовании данной технологии осуществляются через специальный интерфейс, в то время как доступ к реализации модуля (телам подпрограмм и некоторым «внутренним» переменным) запрещен. Эту технологию поддерживают современные версии языков Pascal и С (C++), языки Ада и Modula.

Использование модульного программирования существенно упростило разработку программного обеспечения несколькими программистами. Теперь каждый из них мог разрабатывать свои модули независимо, обеспечивая взаимодействие модулей через специально оговоренные межмодульные интерфейсы. Кроме того, модули в дальнейшем без изменений можно было использовать в других разработках, что повысило производительность труда программистов.

Практика показала, что структурный подход в сочетании с модульным программированием позволяет получать достаточно надежные программы, размер которых не превышает 100 000 операторов. Узким местом модульного программирования является то, что ошибка в интерфейсе при вызове подпрограммы выявляется только при выполнении программы (из-за раздельной компиляции модулей обнаружить эти ошибки раньше невозможно). При увеличении размера программы обычно возрастает сложность межмодульных интерфейсов, и с некоторого момента предусмотреть взаимовлияние отдельных частей программы становится практически невозможно. Для разработки программного обеспечения большого объема было предложено использовать объектный подход.

***Третий этап - объектный подход к программированию*** (с середины 80-х до конца 90-х годов XX в.). Объектно-ориентированное программирование определяется как технология создания сложного программного обеспечения, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного типа (класса), а классы образуют иерархию с наследованием свойств. Взаимодействие программных объектов в такой системе осуществляется путем передачи сообщений (рисунок 1.6).

Объектная структура программы впервые была использована в языке имитационного моделирования сложных систем Simula, появившемся еще в 60-х годах XX в. Естественный для языков моделирования способ представления программы получил развитие в другом специализированном языке моделирования - языке Smalltalk (70-е годы XX в.), а затем был использован в новых версиях универсальных языков программирования, таких, как Pascal, C++, Modula, Java.

Основным достоинством объектно-ориентированного программирования по сравнению с модульным программированием является «более естественная» декомпозиция программного обеспечения, которая существенно облегчает его разработку. Это приводит к более полной локализации данных и интегрированию их с подпрограммами обработки, что позволяет вести практически независимую разработку отдельных частей (объектов) программы. Кроме этого, объектный подход предлагает новые способы организации программ, основанные на механизмах наследования, полиморфизма, композиции, наполнения. Эти механизмы позволяют конструировать сложные объекты из сравнительно простых. В результате существенно увеличивается показатель повторного использования кодов и появляется возможность создания библиотек классов для различных применений.

Бурное развитие технологий программирования, основанных на объектном подходе, позволило решить многие проблемы. Так были созданы среды, поддерживающие визуальное программирование, например, Delphi, C++ Builder, Visual C++, Visual C# и т. д. При использовании визуальной среды у программиста появляется возможность проектировать некоторую часть, например, интерфейсы будущего продукта, с применением визуальных средств добавления и настройки специальных библиотечных компонентов. Результатом визуального проектирования является заготовка будущей программы, в которую уже внесены соответствующие коды.

Использование объектного подхода имеет много преимуществ, однако его конкретная реализация в объектно-ориентированных языках программирования, таких, как Pascal и C++, имеет существенные недостатки:

• фактически отсутствуют стандарты компоновки двоичных результатов компиляции объектов в единое целое даже в пределах одного языка программирования: компоновка объектов, полученных разными компиляторами C++ в лучшем случае проблематична, что приводит к необходимости разработки программного обеспечения с использованием средств и возможностей одного языка программирования высокого уровня и одного компилятора, а значит, требует одного языка программирования высокого уровня и одного компилятора, а значит, требует наличия исходных кодов используемых библиотек классов;

• изменение реализации одного из программных объектов, как минимум, связано с перекомпиляцией соответствующего модуля и перекомпоновкой всего программного обеспечения, использующего данный объект.

Таким образом, при использовании этих языков программирования сохраняется зависимость модулей программного обеспечения от адресов экспортируемых полей и методов, а также структур и форматов данных. Эта зависимость объективна, так как модули должны взаимодействовать между собой, обращаясь к ресурсам друг друга. Связи модулей нельзя разорвать, но можно попробовать стандартизировать их взаимодействие, на чем и основан компонентный подход к программированию.

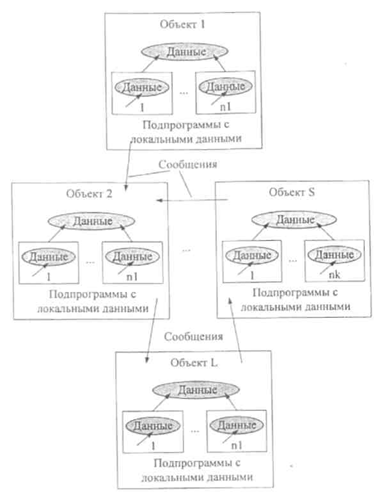


Рисунок 1.6 – Архитектура программы при объектно-ориентированном подходе

***Четвертый этап - компонентный подход и CASE-технологии*** (с середины 90-х годов XX в. до нашего времени).

Компонентный подход предполагает построение программного обеспечения из отдельных компонентов физически отдельно существующих частей программного обеспечения, которые взаимодействуют между собой через стандартизованные двоичные интерфейсы. В отличие от обычных объектов объекты-компоненты можно собрать в динамически вызываемые библиотеки или исполняемые файлы, распространять в двоичном виде (без исходных текстов) и использовать в любом языке программирования, поддерживающем соответствующую технологию. На сегодня рынок объектов стал реальностью, так в Интернете существуют узлы, предоставляющие большое количество компонентов, рекламой компонентов забиты журналы. Это позволяет программистам создавать продукты, хотя бы частично состоящие из повторно использованных частей, т.е. использовать технологию, хорошо зарекомендовавшую себя в области проектирования аппаратуры.

Компонентный подход лежит в основе технологий, разработанных на базе COM (Component Object Model - компонентная модель объектов), и технологии создания распределенных приложений CORBA (Common Object Request Broker Architecture - общая архитектура с посредником обработки запросов объектов). Эти технологии используют сходные принципы и различаются лишь особенностями их реализации.

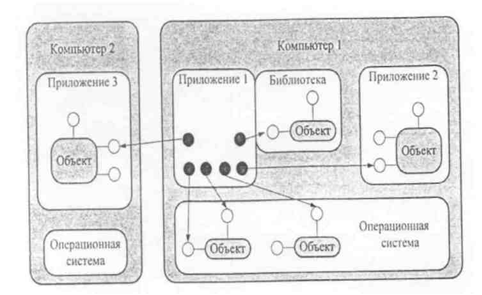


Рисунок 1.7 – Взаимодействие программных компонентов различных типов

Технология СОМ фирмы Microsoft является развитием технологии OLE I (Object Linking and Embedding - связывание и внедрение объектов), которая использовалась в ранних версиях Windows для создания составных документов. Технология СОМ определяет общую парадигму взаимодействия программ любых типов: библиотек, приложений, операционной системы, т. е. позволяет одной части программного обеспечения использовать функции (службы), предоставляемые другой, независимо от того, функционируют ли эти части в пределах одного процесса, в разных процессах на одном компьютере или на разных компьютерах (рисунок 1.7). Модификация СОМ, обеспечивающая передачу вызовов между компьютерами, называется DCOM (Distributed COM - распределенная СОМ).

По технологии СОМ приложение предоставляет свои службы, используя специальные объекты - объекты СОМ, которые являются экземплярами классов СОМ. Объект СОМ так же, как обычный объект включает поля и методы, но в отличие от обычных объектов каждый объект СОМ может реализовывать несколько интерфейсов, обеспечивающих доступ к его полям и функциям. Это достигается за счет организации отдельной таблицы адресов методов для каждого интерфейса (по типу таблиц виртуальных методов). При этом интерфейс обычно объединяет несколько однотипных функций. Кроме того, классы СОМ поддерживают наследование интерфейсов, но не поддерживают наследования реализации, т. е. не наследуют код методов, хотя при необходимости объект класса-потомка может вызвать метод родителя.

Каждый интерфейс имеет имя, начинающееся с символа «I» и глобальный уникальный идентификатор IID (Interface IDentifier). Любой объект СОМ обязательно реализует интерфейс IUnknown (на схемах этот интерфейс всегда располагают сверху). Использование этого интерфейса позволяет получить доступ к остальным интерфейсам объекта.

Объект всегда функционирует в составе сервера - динамической библиотеки или исполняемого файла, которые обеспечивают функционирование объекта. Различают три типа серверов:

• внутренний сервер - реализуется динамическими библиотеками, которые подключаются к приложению-клиенту и работают в одном с ними адресном пространстве - наиболее эффективный сервер, кроме того, он не требует специальных средств;

• локальный сервер — создается отдельным процессом (модулем, ехе), который работает наодном компьютере с клиентом;

• удаленный сервер - создается процессом, который работает на другом компьютере.

Например, Microsoft Word является локальным сервером. Он включает множество объектов, которые могут использоваться другими приложениями.

Для обращения к службам клиент должен получить указатель на соответствующий интерфейс. Перед первым обращением к объекту клиент посылает запрос к библиотеке СОМ, хранящей информацию обо всех, зарегистрированных в системе классах СОМ объектов, и передает ей имя класса, идентификатор интерфейса и тип сервера. Библиотека запускает необходимый сервер, создает требуемые объекты и возвращает указатели на объекты и интерфейсы. Получив указатели, клиент может вызывать необходимые функции объекта.

Взаимодействие клиента и сервера обеспечивается базовыми механизмами СОМ или DCOM, поэтому клиенту безразлично местонахождение объекта. При использовании локальных и удаленных серверов в адресном пространстве клиента создается proxy-объект - заместитель объекта СОМ, а в адресном пространстве сервера СОМ - заглушка, соответствующая клиенту. Получив задание от клиента, заместитель упаковывает его параметры и, используя службы операционной системы, передает вызов заглушке. Заглушка распаковывает задание и передает его объекту СОМ. Результат возвращается клиенту в обратном порядке.

На базе технологии СОМ и ее распределенной версии DCOM были разработаны компонентные технологии, решающие различные задачи разработки программного обеспечения.

OLE-automation или просто Automation (автоматизация) — технология создания программируемых приложений, обеспечивающая программируемый доступ к внутренним службам этих приложений. Вводит понятие диспинтерфейса (dispinterface) - специального интерфейса, облегчающего вызов функций объекта. Эту технологию поддерживает, например, Microsoft Excel, предоставляя другим приложениям свои службы.

ActiveX - технология, построенная на базе OLE-automation, предназначена для создания программного обеспечения как сосредоточенного на одном компьютере, так и распределенного в сети. Предполагает использование визуального программирования для создания компонентов -элементов управления ActiveX. Полученные таким образом элементы управления можно устанавливать на компьютер дистанционно с удаленного сервера, причем устанавливаемый код зависит от используемой операционной системы. Это позволяет применять элементы управления ActiveX в клиентских частях приложений Интернет.

Основными преимуществами технологии ActiveX, обеспечивающими ей широкое распространение, являются:

• быстрое написание программного кода - поскольку все действия, связанные с организацией взаимодействия сервера и клиента берет на программное обеспечение СОМ, программирование сетевых приложений становится похожим на программирование для отдельного компьютера;

• открытость и мобильность - спецификации технологии недавно были переданы в Open Group как основа открытого стандарта;

• возможность написания приложений с использованием знакомых средств разработки, например, Visual Basic, Visual C++, Borland Delphi, Borland C++ и любых средств разработки на Java;

• большое количество уже существующих бесплатных программных элементов ActiveX (к тому же, практически любой программный компонент OLE совместим с технологиями ActiveX и может применяться без модификаций в сетевых приложениях);

• стандартность - технология ActiveX основана на широко используемых стандартах Internet (TCP/IP, HTML, Java), с одной стороны, и стандартах, введенных в свое время Microsoft и необходимых для сохранения совместимости (COM, OLE).

MTS (Microsoft Transaction Server - сервер управления транзакциями) технология, обеспечивающая безопасность и стабильную работу распределенных приложений при больших объемах передаваемых данных.

MIDAS (Multitier Distributed Application Server - сервер многозвенных распределенных приложений) - технология, организующая доступ к данным разных компьютеров с учетом балансировки нагрузки сети.

Все указанные технологии реализуют компонентный подход, заложенный в СОМ. Так, с точки зрения СОМ элемент управления ActiveX - внутренний сервер, поддерживающий технологию OLE-automation. Для программиста же элемент ActiveX - «черный ящик», обладающий свойствами, методами и событиями, который можно использовать как строительный блок при создании приложений.

Технология CORBA, разработанная группой компаний ОМС (Object Management Group - группа внедрения объектной технологии программирования), реализует подход, аналогичный СОМ, на базе объектов и интерфейсов CORBA. Программное ядро CORBA реализовано для всех основных аппаратных и программных платформ и потому эту технологию можно использовать для создания распределенного программного обеспечения в гетерогенной (разнородной) вычислительной среде. Организация взаимодействия между объектами клиента и сервера в CORBA осуществляется с помощью специального посредника, названного VisiBroker, и другого специализированного программного обеспечения.

Отличительной особенностью современного этапа развития технологии программирования, кроме изменения подхода, является создание и внедрение автоматизированных технологий разработки и сопровождения программного обеспечения, которые были названы CASE-технологиями (Computer-Aided Software/System Engineering - разработка программного обеспечения/программных систем с использованием компьютерной поддержки). Без средств автоматизации разработка достаточно сложного программного обеспечения на настоящий момент становится трудно осуществимой: память человека уже не в состоянии фиксировать все детали, которые необходимо учитывать при разработке программного обеспечения. На сегодня существуют CASE-технологии, поддерживающие как структурный, так и объектный (в том числе и компонентный) подходы к программированию.

Появление нового подхода не означает, что отныне все программное обеспечение будет создаваться из программных компонентов, но анализ существующих проблем разработки сложного программного обеспечения показывает, что он будет применяться достаточно широко.

***Технология программирования*** — совокупность методов и средств, применяемых в процессе разработки программного обеспечения.

***Программа*** (program, routine) — упорядоченная последовательность команд (инструкций) компьютера для решения задачи.

***Программное обеспечение*** (software) — совокупность программ обработки данных и необходимых для их эксплуатации документов.

***Задача*** (problem, task) — проблема, подлежащая решению.

***Приложение*** (application) — программная реализация на компьютере решения задачи.

Термин «задача» в программировании означает единицу работы вычислительной системы, требующую выделения вычислительных ресурсов (процессорного времени, памяти).

Процесс создания программ можно представить как последовательность следующих действий:

1. постановка задачи;

2. алгоритмизация решения задачи;

3. программирование.

***Постановка задачи*** (problem definition) — это точная формулировка решения задачи на компьютере с описанием входной и выходкой информации.

***Алгоритм*** — система точно сформулированных правил, определяющая процесс преобразования допустимых исходных данных (выходной информации) в желаемый результат (выходную информацию) за конечное число шагов.

***Программирование*** (programming) — теоретическая и практическая деятельность, связанная с созданием программ.

**По отношению к ПО компьютерные пользователи делятся на следующие группы:**

1. *системные программисты*. Занимаются разработкой, эксплуатацией и сопровождением системного программного обеспечения;

2. *прикладные программисты*. Осуществляют разработку и отладку программ для решения различных прикладных задач;

3. *конечные пользователи*. Имеют элементарные навыки работы с компьютером и используемыми ими прикладными программами;

4. *администраторы сети*. Отвечают за работу вычислительных сетей;

5. *администраторы баз данных*. Обеспечивают организационную поддержку базы данных.

***Сопровождение программы*** — поддержка работоспособности программы, переход на ее новые версии, внесения изменений, исправление ошибок и т.д.

**Основные характеристики программ:**

1. алгоритмическая сложность;

2. состав функций обработки информации;

3. объем файлов, используемых программой;

4. требования к операционной системе (ОС) и техническим средствам обработки, в том числе объем дисковой памяти, размер оперативной памяти дли запуска программы, тип процессора, версия ОС, наличие вычислительной сети и т. д.

**Показатели качества программы:**

1. *мобильность* (многоплатформенность) — независимость от технического комплекса системы обработки данных, ОС, сетевых возможностей, специфики предметной области задачи и т.д.

2. *надежность* — устойчивость, точность выполнения предписанных функций обработки, возможность диагностики возникающих ошибок в работе программы;

3. *эффективность* как с точки зрения требований пользователя, так и расхода вычислительных ресурсов;

4. *учет человеческого фактора* — дружественный интерфейс, контекстно-зависимая подсказка, хорошая документация;

5. *модифицируемость* — способность к внесению изменений, например, расширение функций обработки, переход на другую техническую базу обработки и т. п.

6. *коммуникативность* — максимально возможная интеграция с другими программами, обеспечение обмена данными между программами.

Все программы по характеру использования и категориям пользователей можно разделить на два класса — утилитарные программы и программные продукты.

***Утилитарные программы*** («программы для себя») предназначены для удовлетворения нужд их разработчиков. Чаще всего такие программы выполняют роль отладочных приложений, являются программами решения задач, не предназначенных для широкого распространения.

***Программные продукты*** (изделия) используются для удовлетворения потребностей пользователей, широкого распространения и продажи.

В настоящее время существуют и другие варианты легального распространения программных продуктов, которые появились с использованием глобальных телекоммуникации;

• freeware – бесплатные программы, свободно распространяемые, поддерживаются самим пользователем, который правомочен вносить в них необходимые изменения;

• shareware – некоммерческие (условно-бесплатные) программы, которые могут использоваться, как правило, бесплатно.

Ряд производителей использует OEM-программы (Original Equipment Manufacturer), т. е. встроенные программы, устанавливаемые на компьютеры или поставляемые вместе с компьютерами.

Программный продукт должен быть соответствующим образом подготовлен к эксплуатации (отлажен), иметь необходимую техническую документацию, предоставлять сервис и гарантию надежной работы программы, иметь товарный знак изготовителя, а также наличие кода государственной регистрации.

**Классификация программного обеспечения**

**Можно выделить три класса ПО:**

1. *системное;*

2. *пакеты прикладных программ* (прикладное ПО);

3. *инструментарий технологии программирования* (инструментальные средства для разработки ПО).

**Системное ПО направлено:**

• на создание операционной среды функционирования других программ;

• обеспечение надежной и эффективной работы самого компьютера и вычислительной сети;

• проведение диагностики и профилактики аппаратуры компьютера и вычислительных сетей;

• выполнение вспомогательных технологических процессов (копирование, архивация, восстановление файлов программ и БД и т. п.).

Системное ПО (System Software) – совокупность программ и программных комплексов для обеспечения работы компьютера и вычислительных сетей.

Прикладное ПО служит программным инструментарием решения функциональных задач и является самым многочисленным классом ПО. В данный класс входят программные продукты, выполняющие обработку информации различных предметных областей. Таким образом, прикладное ПО – комплекс взаимосвязанных программ для решения задач определенного класса предметной области.

Инструментарий технологии программирования обеспечивает процесс разработки программ и включает специализированное ПО, которое является инструментальным средством разработки. ПО данного класса поддерживает все технологические этапы процесса проектирования, программирования, отладки и тестирования создаваемых программ. Пользователями данного ПО являются системные и прикладные программисты.

**Системное программное обеспечение**

В составе системного ПО можно выделить две составляющие:

1. базовое ПО;

2. сервисное ПО.

**Базовое ПО** – минимальный набор программных средств, обеспечивающих работу компьютера.

**Сервисное ПО** – программы и программные комплексы, которые расширяют возможности базового ПО и организуют более удобную среду работы пользователя.

В базовое ПО входят:

1. операционная система;

2. операционные оболочки (текстовые, графические);

3. сетевая операционная система.

Компьютер выполняет действия в соответствии с предписаниями программы, созданной на одном из языков программирования. При работе пользователя на компьютере часто возникает необходимость выполнять операции с прикладной программой в целом, организовать работу внешних устройств, проверить работу различных блоков, скопировать информацию и т. д. Эти операции используются для работы с любой программой. Поэтому целесообразно из всего многообразия операций, выполняемых компьютером, выделить типовые и реализовать их с помощью специализированных программ, которые следует принять в качестве стандартных средств, поставляемых имеете с аппаратной частью. Программы, организующие работу устройств и не связанные со спецификой решаемой задачи, вошли в состав комплекса программ, названного операционной системой.

**Операционная система** – совокупность программных средств, обеспечивающих управление аппаратной частью компьютера и прикладными программами, а также их взаимодействием между собой и пользователем.

Операционная система образует автономную среду, не связанную ни с одним из языков программирования. Любая прикладная программа связана с ОС и может эксплуатироваться только на компьютерах, где имеется аналогичная системная среда. Программа, созданная в среде одной ОС, не функционирует в среде другой ОС, если в ней не обеспечена возможность конвертации (преобразования) программ.

Для работы с ОС необходимо овладеть языком этой среды – совокупностью команд, структура которых определяется синтаксисом этого языка.

**ОС выполняет следующие функции:**

• управление работой каждого блока ПО и их взаимодействием;

• управление выполнением программ;

• управление выполнением программ;

• организацию хранения информации во внешней памяти;

• взаимодействие пользователя с компьютером (поддержку интерфейса пользователя).

Обычно ОС хранится на жестком диске, а при его отсутствии выделяется специальный гибкий диск, который называется системным диском. При включении компьютера ОС автоматически загружается с диска в оперативную память и занимает в ней определенное место. ОС создается не для отдельной модели компьютера, а для серии компьютеров, в структуру которых заложена и развивается во всех последующих моделях определенная концепция.

**Операционные системы для ПК делятся:**

• на одно- и многозадачные (в зависимости от числа параллельного выполнения прикладных процессов);

• пользователей, одновременно работающих с ОС);

• непереносимые и переносимые на другие типы компьютеров;

• несетевые и сетевые, обеспечивающие работу в ВС ЭВМ.

**Операционные системы, как правило, сравнивают по следующим критериям:**

• управлению памятью (максимальный объем адресуемого пространства, технические показатели использования памяти);

• функциональным возможностям вспомогательных про

грамм (утилит) в составе ОС;

• наличию компрессии магнитных дисков;

• возможности архивирования файлов;

• поддержке многозадачного режима работы;

• наличию качественной документации;

• условию и сложности процесса инсталляции.

**Наиболее распространенными ОС** для персональных компьютеров в настоящее время являются:

• Windows 7 (Microsoft);

• Windows 8 (Microsoft);

• UNIX;

• OS/2 (IBM);

• Linux.

ОС OS/2 разработана IBM для ПК на основе системной прикладной архитектуры, ранее используемой для больших ЭВМ. Это многозадачная, многопользовательская ОС, обеспечивающая как текстовый, так и графический интерфейс пользователя. OS/2 обеспечивает:

• поддержку графического интерфейса пользователя;

• одновременную обработку нескольких приложений;

• многопоточную обработку нескольких задач одного приложения;

• 32-разрядную обработку данных;

• сжатие данных при записи на магнитный диск;

• защиту памяти.

Важными особенностями OS/2 является высокопроизводительная файловая система HPFS (High Performance File System), имеющая преимущества для серверов БД (в отличие от MS DOS поддерживаются длинные имена файлов), а также поддержка мультипроцессорной обработки — до 16 процессоров типа Intel и Power PC. OS/2 Warp имеет встроенный доступ в Интернет, систему распознавания речи VoiceType, интегрированную версию Lotus Notes Mail для передачи электронной почты. В OS/2 могут выполняться прикладные программы Windows.

ОС UNIX, созданная корпорацией Bell Laboratory, реализует принцип открытых систем и широкие возможности по объединению в составе одной вычислительной системы разнородных технических и программных средств. UNIX обладает следующими свойствами:

• переносимость прикладных программ с одного компьютера на другой;

• поддержка распределенной обработки данных в сети ЭВМ;

• сочетаемость с процессорами RISC.

UNIX получила распространение дли суперкомпьютеров, работающих станций и профессиональных ПК, имеет большое число версий, разработанных различными производителями ПО.

**Сетевая ОС** – это комплекс программ, обеспечивающий обработку, передачу и хранение данных в сети. Сетевая ОС предоставляет пользователям различные вилы сетевых служб {управление файлами, электронная почта, процессы управления сетью и т. д.), поддерживает работу в абонентских системах. Сетевые ОС используют архитектуру «клиент – сервер» или одноранговую архитектуру. Вначале эти ОС поддерживали только ЛВС, сейчас они распространяются и на объединения ЛВС. Наибольшее распространение имеют семейства Windows, NetWare.

Сетевая операционная система семейства Windows является многозадачной, предназначенной для архитектуры «клиент – сервер» и использования различных протоколов транспортного уровня сетевой ОС (IPX/SPX, TCP/IP, Net Bios), имеет 64-разрядную архитектуру и обеспечивает следующие функции локальной сети:

• возможность каждой абонентской системы в сети быть сервером или клиентом;

• совместную работу группы пользователей;

• адресацию оперативной и внешней памяти большого раз-

• многозадачность и многопоточность обработки данных;

• поддержку мультипроцессорной обработки и др.

**Операционные оболочки** – специальные программы, предназначенные для облегчения общения пользователя с командами ОС. Операционные оболочки имеют текстовый и графический варианты интерфейса конечного пользователя. Наиболее популярны для операционной системы MS DOS являлись такие текстовые оболочки как Norton Commander, DOS Navigator, Volkov Commander и др. Эти программы существенно упрощают задание управляющей информации дли выполнения команд ОС, уменьшают сложность работы конечного пользователя.

Среди графических оболочек MS DOS можно было выделить Windows 3.1, Windows 3.11 For Work Group, которые расширяли набор основных (диспетчер файлов, графический редактор, текстовый редактор и т. д.) и сервисных функций, обеспечивающих создание пользователю систем обработки информации в составе одноранговых локальных сетей.

**Расширением базового ПО компьютера является набор сервисных, дополнительно устанавливаемых программ, которые можно классифицировать по функциональному признаку следующим образом:**

• программы диагностики работоспособности компьютера;

• антивирусные программы, обеспечивающие защиту компьютера, обнаружение и восстановление зараженных файлов;

• программы обслуживания дисков, обеспечивающие проверку качества поверхности магнитного диска, контроль сохранности файловой системы на логическом и физическом уровнях, сжатие дисков, резервное копирование данных на внешние носители и др.;

• программы архивирования данных, которые обеспечивают процесс сжатия информации с целью уменьшения объема памяти для ее хранения;

• программы обслуживания сети.

Эти программы часто называют утилитами.

**Утилиты** – программы, служащие для выполнения вспомогательных операций обработки данных или обслуживания компьютеров.

Наибольшее распространение имеют следующие утилиты:

• Norton Utilities;

• программа резервного копировании Acronis True Image for PC.

Антивирусные программы – обнаружение зараженных файлов c возможным "лечением".

Инструментарий технологии программирования

**Инструментарий технологии программирования** — это программные продукты, предназначенные для поддержки технологии программирования (рисунок 1.8).



Рисунок 1.8 – Инструментарий технологии программирования

*Средства для создания приложений* – совокупность языков и систем программирования, инструментальные среды пользователя, а также различные программные компоненты для отладки и поддержки создаваемых программ.

*Язык программирования* – это формализованный язык для описания алгоритма решения задач на компьютере. Языки программирования можно условно разделить на следующие классы:

• *машинные языки* – это языки, воспринимаемые аппаратной частью компьютера (машинные коды);

• *машинно-ориентированные языки*, отражающие структуру конкретного типа компьютера (ассемблер);

• процедурно-ориентированные языки – это языки, в которых имеется возможность описания программы как совокупности процедур, или подпрограмм (Си, Паскаль и др.);

• *проблемно-ориентированные языки*, предназначенные для решения задач определенного класса (ЛИСП, ПРОЛОГ).

Другой классификацией языков является их деление на языки, ориентированные на реализацию основ структурного программирования, основанного на модульной структуре программного продукта и типовых управляющих структурах алгоритмов обработки данных различных программных модулей, и объектно-ориентированные языки, поддерживающие понятие объектов, их свойств и методов обработки.

**Системы программирования включают:**

• *компилятор (транслятор)*;

• *интегрированную среду разработки программ* (не всегда);

• *отладчик;*

• *средства оптимизации кода программ*; - набор библиотек;

• *редактор связей*;

• *сервисные средства* (утилиты) (для работы с библиотеками, текстовыми и двоичными файлами);

• *справочные системы*;

• систему поддержки и управления продуктами программного комплекса.

**Компилятор** транслирует всю программу без се выполнения Трансляторы (интерпретаторы) выполняют пооперационную обработку и выполнение программы.

**Отладчики** (debugger) — специальные программы, предназначенные для трассировки и анализа выполнения других программ. **Трассировка** — это обеспечение выполнения в пооператорном варианте.

**Инструментальная среда пользователя** – это специальные средства, встроенные в пакеты прикладных программ, такие как:

• библиотека функций, процедур, объектов и методов обработки;

• макрокоманды;

• клавишные макросы;

• языковые макросы;

• конструкторы экранных форм и объектов;

• генераторы приложений;

• языки запросов высокого уровня;

• конструкторы меню и др.

Интегрированные среды разработки программ объединяют набор средств для их комплексного применения на технологических этапах создании программы.

Средства для создания информационных систем (ИС) и технологий поддерживают полный цикл проектирования сложной информационной системы или технологии от исследования объекта автоматизации до оформления проектной и прочей документации на информационную систему или технологию. Они позволяют вести коллективную работу нал проектом за счет возможности работы в локальной сети, экспорта-импорта любых фрагментов проекта, организации управления проектом.

Одним из современных средств разработки ИС является CASE-технология (CASE — Computer-Aided System Engineering) – программный комплекс, автоматизирующий весь технологический процесс анализа, проектирования, разработки и сопровождения сложных программных систем.

**Средства CASE-технологий делятся:**

• *на встроенные в систему реализации* – нее решения по проектированию и реализации привязки к выбранной СУБД;

• *независимые от системы реализации* – все решении по проектированию ориентированы на унификацию (определение) начальных этапов жизненного цикла программы и средств их документирования, обеспечивают большую гибкость в выборе средств реализации.

Основное достоинство CASE-технологии – это поддержка коллективной работы над проектом за счет возможности работы в локальной сети разработчиков, экспорта (импорта) любых фрагментов проекта, организованного управления проектами.

В некоторых CASE-системах поддерживается кодогенерация программ – создание каркаса программ и создание полного продукта.

Примеры программных продуктов для создания приложений: Visual C++, Delphi, Visual Basic и т. д.

**Пакеты прикладных программ**

Классификации пакетов прикладных программ (ППП) приведена на рисунке 1.9,

Проблемно-ориентированные ППП. Для некоторых предметных областей возможна типизация функций управления, структуры данных и алгоритмов обработки. Это вызвало разработку значительного количества ППП одинакового функционального назначения:

• ППП автоматизированного бухгалтерского учета;

• ППП финансовой деятельности;

• ППП управления персоналом;

• ППП управления производством;

• банковские информационные системы и т. п.

**Основные тенденции развития:**

• создание программных комплексов в виде автоматизированных рабочих мест (АРМ) управленческого персонала;

• создание интегрированных систем управления предметной областью на базе вычислительных сетей, объединяющих АРМы;

• организации данных больших информационных систем в виде распределенной БД на сети ЭВМ;

• наличие простых языков средств конечного пользователя и др.

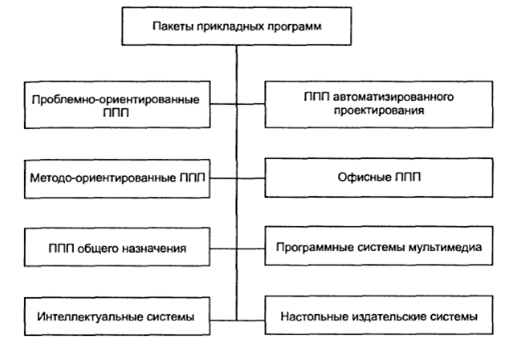


Рисунок 1.9 – Классификация пакетов прикладных программ

ППП автоматизированного проектирования предназначены для поддержки работы конструкторов и технологов, связанных с разработкой чертежей, схем, графическим моделированием и конструированием. Отличительными особенностями этого класса ППП являются высокие требования к аппаратному обеспечению, наличие библиотек встроенных функций, объектов, интерфейсов с графическими системами и БД (AutoCAD).

**К ППП общего назначения относят:**

1. *Системы управления базами данных (СУБД)*, обеспечивающие организацию и хранение локальных БД на автономно работающих компьютерах либо централизованное хранение БД на файл-сервере и сетевой доступ к ним. В современных СУБД (например, MS Access, MS SQL Server, Oracle) содержатся элементы CASE-технологии процесса проектирования, в частности:

• визуализирована схема БД;

• осуществлена автоматическая поддержка целостности БД при различных видах обработки (включение, удаление, модификация);

• предоставляются так называемые мастера, обеспечивающие поддержки процесса проектирования;

• созданы шаблоны (прототипы) структур БД, отчетов, форм

и т.д.

2.  *Серверы БД* — это ПО, предназначенное для создания и использования при работе в сети интегрированных БД в архитектуре "клиент – сервер». Многопользовательские СУБД в сетевом варианте обработки информации хранят данные на файл-сервере, специально выделенном компьютере, но сама обработка ведется на рабочих станциях. Серверы БД в отличие от этого большую часть обработки (хранение, поиск, извлечение и передачу данных клиенту) данных выполняют самостоятельно, одновременно обеспечивая данными большое число пользователей сети. Общим для различных видов серверов БД является использование реляционного языка SQL (Structured Query Language) для реализации запросов к данным. Большинство серверов БД поддерживает несколько платформ, широкий спектр протоколов передачи данных. Проблемы: обеспечение целостности данных, тиражирование данных по узлам сети и синхронное обновление.

3. *Генераторы отчетов* (серверы отчетов), обеспечивающие реализацию запросов и формирование отчетов в печатном или экранном виде в условиях сети с архитектурой "клиент — сервер». Сервер отчетов подключается к серверу БД, использующему драйверы сервиса БД (Crystal Reports, Profit for Windows).

4. *Текстовые процессоры*, предназначенные для работы с текстовыми документами. Развитием данного направления являются издательские системы (Microsoft Word).

5. *Табличные процессоры*, являющиеся удобной средой для вычислений конечным пользователем, содержат средства деловой графики, средства специализированной обработки (Microsoft Excel).

6. *Средства презентационной графики* - специализированные программы, предназначенные для создания изображений и их показа на экране, подготовки слайд-фильмов, мультфильмов и их проектирования (Microsoft PowerPoint).

7. *Интегрированные пакеты набор нескольких программных продуктов*, функционально дополняющих друг друга, поддерживающие единые информационные технологии, реализованные на единой операционной и вычислительной платформе (Microsoft Office). Компоненты интегрированных пакетов могут работать изолированно друг от друга, имеют общий интерфейс, благодаря этому их лучше осваивать.

Методо-ориентированные ППП, Данный класс охватывает программные продукты, обеспечивающие независимо от предметной области и функции информационных систем математические, статистические и другие методы решения задач. Наиболее распространены методы математического программирования, решения дифференциальных уравнений, имитационного моделирования, исследовании операций (Storm, SYSTAT, STATISTICA, SAS и другие).

Офисные ППП. Данный класс охватывает программы, обеспечивающие ориентационное управление деятельностью офиса:

• органайзеры (планировщики) — ПО дли планирования рабочего времени, составления протоколов встреч, расписании, веление записей и телефонной книжки. В состав входят: калькулятор, записная книжка, часы, календарь и т. п.

• программы-переводчики, средства проверки орфографии, распознавание текста (Tiger – система распознавания русскою языка. Stylus Lingvo Office, содержащий Fine Reader, Stylus For Windows — переводчик на указанный язык, корректор Орфографии Lingvo Corrector и резидентный словарь Lingvo),

• коммуникационные пакеты, предназначенные для организации взаимодействия пользователей с удаленными абонентами или информационными ресурсами сети;

• браузеры, средства создания WWW-страниц;

• средства электронной почты (MS Outlook ).

Настольные издательские системы. Данный класс ПО включает программы (PageMaker, CorelDraw, Adobe PhotoShop, QuarkXPress 2015 и т. д.), обеспечивающие информационную технологию компьютерной издательской деятельности:

• форматирование и редактирование текстов;

• автоматическую разбивку текста на страницы;

• компьютерную верстку печатной страницы;

• монтирование графики;

• подготовку иллюстраций и т. п.

Программные средства мультимедиа

Основное назначение данных программных средств – создание и использование аудио- и видеоинформации для расширения информационного пространства пользователя (различные БД компьютерных произведений искусства, библиотеки звуковых записей (и т. д.).

Системы искусственного интеллекта:

• программы-оболочки для создания экспертных систем путем наполнения баз знаний и правил логического вывода;

• готовые экспертные системы для принятия решений в рамках определенных предметных областей;

• системы анализа и распознавания речи, текста и т. п.

Примеры систем искусственного интеллекта: FIDE, MYSIN, Guru, ИНТЕР-ЭКСПЕРТ и др.

**Инструментальное программное обеспечение. Основные понятия и определения**

Очевидно, что большие размеры и высокая сложность разрабатываемых ПС при ограничениях на бюджетные и временные затраты проекта могут привести к низкому качеству конечных программных продуктов и системы в целом. В этой связи в последнее время все большее внимание уделяется современным технологиям и инструментальным средствам, обеспечивающим автоматизацию процессов ЖЦ ПС (CASE-средствам). Использование таких инструментальных средств позволяет существенно сократить длительность и стоимость разработки систем и ПС при одновременном повышении качества процесса разработки и, как следствие, качества разработанных ПС.

В истории развития CASE-средств обычно выделяется шесть периодов. Данные периоды различаются применяемой техникой и методами разработки ПС. Эти периоды используют в качестве инструментальных средств следующие средства.

**Период 1**. Ассемблеры, анализаторы.

**Период 2**. Компиляторы, интерпретаторы, трассировщики.

**Период 3.** Символические отладчики, пакеты программ.

**Период 4**. Системы анализа и управления исходными текстами.

**Период 5**. Первое поколение CASE (CASE-I). Это CASE-средства, позволяющие выполнять поддержку начальных работ процесса разработки ПС и систем (анализ требований к системе, проектирование архитектуры системы, анализ требований к программным средствам, проектирование программной архитектуры, логическое проектирование баз данных). Адресованы непосредственно системным аналитикам, проектировщикам, специалистам в предметной области. Поддерживают графические модели, экранные редакторы, словари данных. Не предназначены для поддержки полного ЖЦ ПС.

Период 6. Второе поколение CASE (CASE-II). Представляют собой, как правило, набор (линейку) инструментальных средств, каждое их которых предназначено для поддержки отдельных этапов процесса разработки или других процессов ЖЦ ПС. В совокупности обычно поддерживают практически полный ЖЦ ПС. Используют средства моделирования предметной области, графического представления требований, поддержки автоматической кодогенерации ПС. Содержат средства контроля и управления разработкой, интеграции системной информации, оценки качества результатов разработки. Поддерживают моделирование и прототипирование системы, тестирование, верификацию, анализ сгенерированных программ, генерацию документации по проекту.

Ко второму поколению CASE-средств относятся, например, линейки Telelogic и AllFusion.

CASE-технологии предлагают новый, основанный на автоматизации подход к концепции ЖЦ ПС. Современные варианты CASE-моделей ЖЦ, называемые обычно RAD-моделями.

Наибольшие изменения в ЖЦ ПС при использовании CASE-технологий касаются первых этапов ЖЦ, связанных с анализом требований и проектированием. CASE-средства позволяют использовать визуальные среды разработки, средства моделирования и быстрого прототипирования разрабатываемой системы или ПС. Это позволяет на ранних этапах разработки оценить, насколько будущая система или программное средство устраивает заказчика и насколько она дружественна будущему пользователю.

Таблица 1.1 содержит усредненные оценки трудозатрат по основным этапам разработки ПС при различных подходах к процессу разработки. Номерам строк в данной таблице соответствуют: 1 – традиционная разработка с использованием классических технологий; 2 – разработка с использованием современных структурных методологий проектирования; 3 – разработка с использованием CASE–технологий.

Таблица 1.1 – Сравнительная оценка трудозатрат по этапам процесса разработки программных средств

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № подхода | Анализ, % | Проектирование, % | Кодирование, % | Тестирование, % |
| 1 | 20 | 15 | 20 | 45 |
| 2 | 30 | 30 | 15 | 25 |
| 3 | 40 | 40 | 5 | 15 |

Из таблицы видно, что при традиционной разработке ПС основные усилия направлены на кодирование и тестирование, а при использовании CASE-технологий – на анализ и проектирование, поскольку CASE предполагают автоматическую кодогенерацию, автоматизированное тестирование и автоматический контроль проекта. Сопровождение кодов ПС заменяется сопровождением спецификаций проектирования. В результате данных факторов цена ошибок, вносимых в проект при разработке и сопровождении ПС и систем, существенно снижается.

Резюме

В истории развития CASE-средств обычно выделяется шесть периодов. При традиционной разработке ПС основные усилия направлены на кодирование и тестирование, при использовании CASE-технологий – на анализ и

проектирование.

**Базовые принципы построения CASE-средств**

Большинство CASE-средств основано на парадигме метод – нотация –средство.

Парадигма – это система изменяющихся форм некоторого понятия. В данном случае метод реализуется с помощью нотаций. Метод и нотации поддерживаются инструментальными средствами.

*Метод* – это систематическая процедура или техника генерации описаний компонент ПС.

*Нотация* – это система обозначений, предназначенная для описания структуры системы, элементов данных, этапов обработки; может включать графы, диаграммы, таблицы, схемы алгоритмов, формальные и естественные языки. Например, метод JSP реализуется с помощью нотации, базирующейся на применении четырех базовых конструкций данных. Современной нотацией методологии SADT является IDEF0.

*Средства* – это инструментарий для поддержки методов, помогающий пользователям при создании и редактировании графического проекта в интерактивном режиме, способствующий организации проекта в виде иерархии уровней абстракции, выполняющий проверки соответствия компонентов. Например, средством, поддерживающим метод JSP, является SmartDraw. IDEF0 поддерживается средством BPwin.

*Фактически CASE-средство* – это совокупность графически ориентированных инструментальных средств, поддерживающих процессы или отдельные этапы процессов ЖЦ ПС и систем.

К CASE-средствам может быть отнесено любое программное средство,

обеспечивающее автоматическую помощь при разработке ПС, их сопровождении или управлении проектом, базирующееся на следующих основополагающих принципах:

1. *Графическая ориентация*. В CASE-средствах используется мощная графика для описания и документирования систем или ПС и для улучшения интерфейса с пользователем.

2. *Интеграция.* CASE-средство обеспечивает легкость передачи данных между своими компонентами и другими средствами, входящими в состав линейки CASE-средств. Это позволяет поддерживать совокупность процессов ЖЦ ПС.

3. Локализация всей проектной информации в репозитории (компьютерном хранилище данных). Исполнителям проекта доступны соответствующие разделы репозитория в соответствии с их уровнем доступа. Это обеспечивает поддержку принципа коллективной работы. Информация из репозитория может использоваться для работ над текущим проектом, в том числе для автоматической кодогенерации ПС или систем, разработки следующих проектов, сбора статистики по выполненным ранее проектам организации.

**Помимо данных принципов в основе концептуального построения CASE-средств лежат следующие положения.**

1. *Человеческий фактор*. Его учет позволяет привести процессы ЖЦ ПС и систем к легкой, удобной и экономичной форме.

2. *Использование базовых программных средств, применяющихся в других приложениях* (СУБД, компиляторы с различных языков программирования, отладчики, языки четвертого поколения 4GL и др.).

3. *Автоматизированная или автоматическая кодогенерация*. При автоматизированной кодогенерации выполняется частичная генерация кодов программного средства, остальные участки программируются вручную. При автоматической кодогенерации выполняется полная генерация кодов программного средства. Возможны различные виды генерации (например, генерация проектной документации, базы данных по информационной модели, кодов из разработанных спецификаций программного средства; автоматическая сборка модулей, хранящихся в репозитории).

4. *Ограничение сложности*. Такое ограничение позволяет поддерживать сложность компонентов разрабатываемого программного средства или системы на уровне, доступном для понимания, использования и модификации.

5. *Доступность для различных категорий пользователей*, в том числе заказчиков, специалистов в предметной области, системных аналитиков, проектировщиков, программистов, тестировщиков, инженеров по качеству, менеджеров проектов. CASE-средства содержат инструменты различного функционального назначения, поддерживающие различные этапы основных, вспомогательных и организационных процессов ЖЦ ПС и систем.

6. *Рентабельность,* обеспечивающая быструю окупаемость денежных средств, вложенных в приобретение CASE-средства, за счет сокращения сроков и стоимости проектов.

7. Сопровождаемость. CASE-средства обладают способностью адаптации к изменяющимся требованиям и целям проекта.

Резюме

CASE-средства представляют собой совокупность графически ориентированных инструментальных средств, поддерживающих ЖЦ ПС и систем. CASE-средства базируются на принципах графической ориентации, интеграции и локализации всей проектной информации в репозитории. В основе построения CASE-средств лежат человеческий фактор, использование базовых ПС, автоматизированная или автоматическая кодогенерация, ограничение сложности, доступность для разных категорий пользователей, рентабельность, сопровождаемость.

**Основные функциональные возможности CASE-средств**

В состав CASE-средств входят четыре основных компонента:

1. *Средства централизованного хранения всей информации о проекте* (репозиторий). Предназначены для хранения информации о разрабатываемом программном средстве или системе в течение всего ЖЦ разработки.

2. *Средства ввода*. Служат для ввода данных в репозиторий, организации взаимодействия участников проекта с CASE-средством. Должны поддерживать различные методологии анализа, проектирования, тестирования, контроля. Предназначены для использования в течение ЖЦ программного средства или системы различными категориями участников проекта (системными аналитиками, проектировщиками, программистами, тестировщиками, менеджерами, специалистами по качеству и т.д.).

3. *Средства анализа и разработки*. Предназначены для анализа различных видов графических и текстовых описаний и их преобразований в процессе разработки.

4. Средства вывода. Служат для кодогенерации, создания различного вида документов, управления проектом.

**Все компоненты CASE-средств в совокупности обладают следующими функциональными возможностями:**

• поддержка графических моделей;

• контроль ошибок;

• поддержка репозитория;

• поддержка основных, вспомогательных и организационных процессов ЖЦ ПС.

*Поддержка графических моделей*

В CASE-средствах разрабатываемые ПС представляются схематически. На разных уровнях проектирования могут использоваться различные виды и нотации графического представления ПС. Обычно применяются диаграммы различных типов, в том числе иерархии требований, диаграммы функционального моделирования (например IDEF0, DFD), диаграммы информационного моделирования (например IDEF1X), структурограммы , диаграммы Джексона, диаграммы Варнье – Орра, UML-диаграммы и т.п.

Разработка диаграмм осуществляется с помощью специальных графических редакторов, основными функциями которых являются создание и редактирование иерархически связанных диаграмм, их объектов и связей между объектами, а также автоматический контроль ошибок.

*Контроль ошибок*

В CASE- средствах , как правило, реализуются следующие типы контроля:

1. Контроль синтаксиса диаграмм и типов их элементов. Например, при IDEF0-моделировании контролируется максимальное и минимальное количество функциональных блоков на диаграммах, наличие дуги управления и выходной дуги для любого функционального блока и т.п.

2. Контроль полноты и корректности диаграмм. При данном типе контроля выполняется проверка наличия имен у всех элементов диаграмм, проверка наличия необходимых описаний в репозитории и др.

3. Контроль декомпозиции функций. При данном типе контроля выполняется оценка декомпозиции на основе различных метрик. Например, может быть оценена эффективность и корректность декомпозиции с точки зрения связности и сцепления модулей.

4. Сквозной контроль диаграмм одного или различных типов на предмет их взаимной корректности. Например, при IDEF0-моделировании контролируется соответствие граничных дуг родительского блока с внешними дугами дочерней диаграммы. При разработке IDEF0- и IDEF1Х-моделей предметной области выполняется контроль их взаимной корректности и непротиворечивости.

*Поддержка репозитория*

Основными функциями репозитория являются хранение, обновление, анализ, визуализация всей информации по проекту и организация доступа к ней. Репозиторий обычно хранит более 100 типов объектов (например, диаграммы, определения экранов и меню, проекты отчетов, описания данных, модели данных, модели обработки, исходные коды, элементы данных).

Каждый информационный объект, хранящийся в репозитории, описывается совокупностью своих свойств, например, идентификатор, тип, текстовое описание, компоненты, область значений, связи с другими объектами, времена создания и последнего обновления объекта, автор и т.п.

Репозиторий является базой для автоматической генерации документации по проекту. Основными типами отчетов являются:

• отчеты по содержимому – включают информацию по потокам данных и их компонентов; списки функциональных блоков диаграмм и их входных и выходных потоков; списки всех информационных объектов и их атрибутов; историю изменений объектов; описания модулей и интерфейсов между ними; планы тестирования модулей и т.п.;

• отчеты по перекрестным ссылкам – содержат информацию по связям всех вызывающих и вызываемых модулей; списки объектов репозитория, к которым имеет доступ конкретный исполнитель проекта; информацию по связям между диаграммами и конкретными данными; маршруты движения данных от входа к выходу;

• отчеты по результатам анализа – включают данные по взаимной корректности диаграмм, списки неопределенных информационных объектов, списки неполных диаграмм, данные по результатам анализа структуры проекта и т.п.;

• отчеты по декомпозиции объектов – включают совокупности объектов, входящих в каждый объект, а также объекты, в состав которых входит каждый объект.

Поддержка процессов жизненного цикла программных средств и систем

**Основой поддержки процесса разработки являются следующие свойства современных CASE-средств.**

1. *Покрытие всего жизненного цикла систем или программных сред*ств. Как отмечалось, современные CASE-средства поддерживают практически полный ЖЦ ПС. Однако первоочередное внимание уделяется начальным работам процесса разработки – анализу требований к системе, проектированию системной архитектуры, анализу требований к программным средствам и проектированию программной архитектуры. Грамотная разработка требований к системе и ПС является основой всего проекта, их полнота и корректность определяют уровень соответствия результатов разработки требованиям заказчика.

2. *Поддержка прототипирования*. Большинство моделей ЖЦ, предназначенных для разработки сложных или критичных продуктов, базируются на применении прототипирования. Это касается в первую очередь моделей, поддерживающих инкрементную и эволюционную стратегии разработки. Прототипирование применяется на ранних этапах ЖЦ и позволяет уточнять требования к системе или программному средству, а также прогнозировать поведение разрабатываемого продукта.

3. *Поддержка современных методологий разработки систем или программных средств.* Современные линейки CASE-средств поддерживают, как правило, различные методологии, предназначенные для использования на различных этапах процесса разработки. При этом выполняется графическая поддержка построения диаграмм различных типов, контроль корректности использования шагов проектирования и подготовка документации.

4. Автоматическая кодогенерация. Кодогенерация позволяет построить автоматически до 90 % исходных кодов на языках высокого уровня. Различными CASE-средствами поддерживаются практически все известные языки программирования.

Средства кодогенерации можно подразделить на два вида:

• средства генерации управляющей структуры продукта; данные средства выполняют автоматическое построение логической структуры программного средства, кодов для базы данных, файлов, экранов, отчетов. Остальные фрагменты программного средства кодируются вручную;

• средства генерации полного продукта; данные средства позволяют на основе разработанных спецификаций или моделей генерировать полные коды программного средства, пользовательскую и программную документацию к нему.

Резюме

В состав CASE-средств входят средства централизованного хранения информации о проекте (репозиторий), средства ввода, средства анализа и разработки, средства вывода. Все компоненты CASE-средств в совокупности поддерживают графические модели, репозиторий, процесс разработки и ряд вспомогательных и организационных процессов, выполняют контроль ошибок.

**Назначение и виды инструментального ПО**

Все CASE-средства подразделяются на типы, категории и уровни.

**Классификация по типам**

**Данная классификация отражает функциональное назначение CASE-средства в ЖЦ ПС и систем.**

1. *Анализ и проектирование*

Средства этого типа используются для поддержки начальных этапов процесса разработки: анализа предметной области, разработки требований к системе, проектирования системной архитектуры, разработки требований к программным средствам, проектирования программной архитектуры, технического проектирования программных средств. Средства данного типа поддерживают известные методологии анализа и проектирования. На выходе генерируются спецификации системы, ее компонентов и интерфейсов, связывающих эти компоненты, архитектура системы, архитектура программного средства, технический проект программного средства, включая алгоритмы и определения структур данных.

К средствам данного типа можно отнести, например, AllFusion Process Modeler (BPwin), CASE.Аналитик, Design/IDEF, Telelogic DOORS, Telelogic Modeler, Telelogic TAU, Telelogic Rhapsody, Telelogic Statemate

2. *Проектирование баз данных и файлов*

Средства этого типа обеспечивают логическое моделирование данных, автоматическое преобразование моделей данных в третью нормальную форму, автоматическую генерацию схем баз данных и описаний форматов файлов на уровне программного кода. К средствам этого типа можно отнести, например, AllFusion Data Modeler (ERwin), CA ERwin Data Model Validator (ранее ERwin Examiner), S-Designor, Silverrun, Designer2000, Telelogic TAU, Telelogic Rhapsody.

3. *Программирование и тестирование*

Средства этого типа поддерживают седьмую работу процесса разработки (программирование и тестирование). Данные средства выполняют автоматическую кодогенерацию ПС на основе спецификаций или моделей. Содержат графические редакторы, средства поддержки работы с репозиторием, генераторы и анализаторы кодов, генераторы тестов, анализаторы покрытия тестами, отладчики.

К средствам данного типа можно отнести, например, TAU/Developer, TAU/Tester, Logiscope Audit, Logiscope RuleChecker, Logiscope TestChecker, Logiscope Reviewer, Rhapsody Developer.

4. *Сопровождение и реинженерия*

Общей целью средств этого типа является поддержка корректировки, изменения, преобразования, реинженерия существующей системы, поддержка документации по проекту. К данным средствам относятся средства документирования, анализаторы программ, средства управления изменениями и конфигурацией ПС и систем, средства реструктурирования и реинженерии (реинженерия, реинженеринг – reverse engineering – обратное проектирование, например,

построение спецификаций или моделей по исходным текстам программ), средства обеспечения мобильности, позволяющие перенести разработанную систему или программные средства в новое операционное или аппаратное окружение.

Средства реинженерии включают:

• статические анализаторы для генерирования схем программного

• средства из его кодов и оценки влияния модификаций;

• динамические анализаторы, включающие трансляторы со встроенными

• отладочными возможностями;

• документаторы, автоматически обновляющие документацию при изменении кода программного средства;

• редакторы кодов, автоматически изменяющие при редактировании кодов предшествующие ему структуры, в том числе и спецификации требований;

• средства доступа к спецификациям, позволяющие выполнять их модификацию и генерацию модифицированного кода;

• средства реверсной инженерии, транслирующие коды в спецификации или модели.

К средствам данного типа можно отнести, например, Telelogic DocExpress, Telelogic Synergy, Telelogic Change, средства линейки AllFusion Change Management Suite.

Следует отметить, что ряд CASE-средств других типов содержат в своем составе средства реинженерии. Это касается, например, CASE-средств AllFu-sion Data Modeler, Telelogic Rhapsody.

5. *Окружение*

К средствам данного типа относятся средства поддержки интеграции CASE-средств и данных. К данному типу можно отнести, например, Telelogic Rhapsody Gateway, Telelogic Rhapsody Interface Pack, AllFusion Data Profiler, AllFusion Model Manager, AllFusion Model Navigator.

6. Управление проектом

К средствам данного типа относятся средства поддержки процесса управления ЖЦ ПС и систем. Их функциями являются планирование, контроль, руководство, организация взаимодействия и т.п. К средствам данного типа можно отнести, например, Telelogic Focal Point, Telelogic Dashboard, AllFusion Process Management Suite, ADvisor.

Резюме

Классификация CASE-средств по типам отражает функциональное назначение CASE-средства в ЖЦ ПС. Выделяют типы CASE-средств, ориентированные на следующие этапы процесса разработки и другие процессы ЖЦ: анализ и проектирование, проектирование баз данных и файлов, программирование и тестирование, сопровождение и реинженерия, окружение, управление проектом.

Классификация по категориям

**Данная классификация отражает уровень интегрированности CASE-средств по выполняемым функциям.**

1. *Категория Tool* (tool – рабочий инструмент)

Включает средства самого низкого уровня интегрированности. В данную категорию средств входят инструментальные средства, решающие небольшую автономную задачу при разработке программного средства или системы. Обычно средства данной категории являются компонентами CASE-средств более высокого уровня интегрированности.

2. *Категория ToolКit* (toolкit – набор инструментов, пакет разработчика) Включает CASE-средства среднего уровня интегрированности. Средства данной категории используют репозиторий для всей информации о проекте и ориентированы обычно на поддержку одного этапа или одной работы процесса разработки или на поддержку одного из вспомогательных или организационных процессов ЖЦ ПС или систем. CASE-средства данной категории представляют собой интегрированную совокупность инструментальных средств, имеющих как правило общую функциональную ориентацию.

К CASE-средствам данной категории может быть отнесено, например, большинство CASE-средств из линеек Telelogic и AllFusion при их изолированном использовании.

3. Категория Workbench (workbench – рабочее место). CASE-средства данной категории обладают самой высокой степенью интеграции. Они представляют собой интегрированную совокупность инструментальных средств, поддерживающих практически весь процесс разработки и ряд вспомогательных и организационных процессов ЖЦ ПС и систем. Используют репозиторий для хранения информации по проекту, поддерживают организацию коллективной работы над проектом.

Обычно к категории Workbench относятся линейки CASE-средств при их интегральном использовании. Примерами являются линейки Telelogic и AllFusion. Данные линейки CASE-средств поддерживает практически полностью процесс разработки ПС и систем, процессы сопровождения, документирования, управления конфигурацией, частично процессы обеспечения качества, верификации, аттестации. Таким образом, линейки Telelogic и AllFusion поддерживают практически весь ЖЦ ПС и систем.

Резюме

Классификация по категориям отражает уровень интегрированности CASE-средств по выполняемым функциям. Различают категории Tool, ToolКit, Workbench.

Классификация по уровням

**Данная классификация связана с областью действия CASE-средств в ЖЦ ПС, систем и организаций.**

1. *Верхние (Upper) CASE-средства*

CASE-средства данного уровня называют средствами компьютерного планирования. Их основной целью является помощь руководителям организаций, предприятий и конкретных проектов в определении политики организации и создании планов проекта. CASE-средства данного уровня позволяют строить модель предметной области, проводить анализ различных сценариев (существующего, наилучших, наихудших), накапливать информацию для принятия оптимальных решений. Таким образом, применительно к ЖЦ ПС и систем данные средства поддерживают процесс заказа и первую работу процесса разработки (подготовка процесса разработки). Графические средства данного уровня используются как формализованный язык общения между заказчиком (пользователем) и разработчиком требований.

К средствам данного уровня можно отнести, например, Telelogic System Architect, Telelogic Focal Point, Telelogic Dashboard, средства линейки AllFusion Modeling Suite.

2. *Средние (Middle) CASE-средства*

CASE-средства данного уровня поддерживают начальные этапы процесса разработки (анализ предметной области, разработка требований к системе, проектирование системной архитектуры, разработка требований к программным средствам, проектирование программной архитектуры). При этом встроенные графические средства используются как формализованный язык общения между заказчиком (пользователем) и разработчиком спецификаций требований.

Обычно данные средства обладают возможностями накопления и хранения информации по проекту. Это позволяет использовать накопленные данные как в текущем, так и в других проектах. Например, с помощью накопленной информации могут оцениваться продукты текущего проекта. При этом аналогичная информация предыдущих проектов используется в качестве базовой для оценки.

CASE-средства данного уровня зачастую поддерживают прототипирование и автоматическое документирование.

К CASE-средствам данного уровня можно отнести, например, линейку AllFusion Modeling Suite, средства Telelogic DOORS, Telelogic Modeler, Telelogic Tau, Telelogic Rhapsody, Telelogic Statemate, Telelogic DocExpress.

3. Нижние (Lower) CASE-средства

CASE-средства данного уровня поддерживают вторую половину работ процесса разработки ПС. Содержат графические средства, исключающие необходимость разработки физических мини – спецификаций для программных модулей. Спецификации представляются обычно в виде моделей, которые непосредственно преобразуются в программные коды разрабатываемого программного средства или системы. Автоматически генерируется до 90 % кодов. Входной информацией для кодогенераторов являются спецификации, разработанные как в CASE-средствах данного уровня, так и в CASE-средствах среднего уровня.

CASE-средства нижнего уровня, как правило, поддерживают также прототипирование, тестирование, управление конфигурацией, генерацию документации, облегчают модификацию и сопровождение ПС или систем.

К CASE-средствам данного уровня можно отнести AllFusion Data Modeler, Telelogic Rhapsody, Telelogic Tau, Telelogic Statemate, Telelogic TAU Logiscope, Telelogic Change, Telelogic Synergy, Telelogic DocExpress.

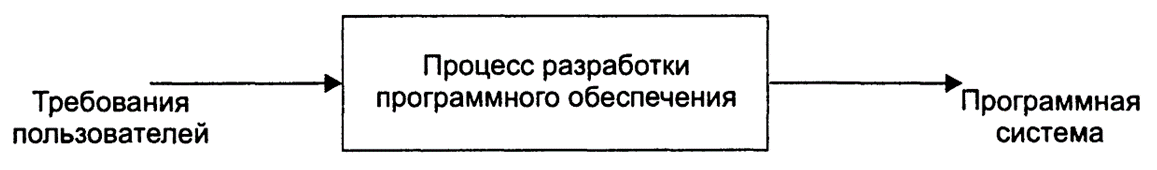
Следует отметить, что в состав CASE-средств среднего и высокого уровней интегрированности обычно входят инструментальные средства, относящиеся к нескольким уровням. Линейки CASE-средств, предназначенные для поддержки всего ЖЦ ПС и систем, включают в свой состав средства всех трех уровней.

Резюме

Классификация по уровням связана с областью действия CASE-средств в ЖЦ ПС и систем. Различают верхние, средние и нижние CASE-средства. Линейки CASE-средств включают в свой состав средства всех трех уровней.

1.3 **Модели процесса разработки программного обеспечения (Презентация 2)**

Процесс разработки программного обеспечения



**Жизненный цикл ПО** это период времени, который начинается с момента принятия решения о необходимости создания программного продукта и заканчивается в момент его полного изъятия из эксплуатации.

Этот цикл включает процесс построения и развития ПО.

1. Анализ предметной области и формирование требований (концепций)
2. Проектирование
3. Реализация
4. Тестирование
5. Ввод в эксплуатацию
6. Эксплуатация (сопровождение проекта)

**Модели процессов разработки ПО**

Модель процесса (МП) описывает обобщенный процесс, и обычно включает:

набор этапов, на которые должен быть разделен процесс;

порядок, в котором эти процессы должны выполняться;

разные ограничения и условия на выполнение этих этапов.

Основное предположение для модели процесса заключается в том, что в тех ситуациях, когда данная модель является применимой, ее использование в качестве процесса проекта позволит:

1. уменьшить расходы,
2. повысить качество,
3. уменьшить время разработки,
4. а также принести другие выгоды.

Другими словами, модель процесса предоставляет общие методические указания для разработки подходящего процесса выполнения проекта.

**Версии (version) ПО**

Выходная версия продукта (release)

A -мажорная версия (major version) ПО

B -минорная версия (minor version) ПО

C –релиз (releas) ПО

D –сборка (build) ПО

**Готовые к поставке ПО должны быть классифицированы следующим образом:**

* ES – поставка - поставка прототипа
* PA – поставка – поставка альфа-версии
* PB - поставка - поставка бета-версии
* RP – поставка – окончательная поставка