|  |  |
| --- | --- |
| лого-РГСУ-2015.png | **Российский государственный социальный университет**  **Факультет информационных технологий** |

Практическое задание № 2

**по дисциплине «Технология программирования»**

|  |  |
| --- | --- |
| **ФИО студента** | Тарасова Андрей Валерьевич |
| **Направление подготовки** | Программная инженерия |
| **Группа** | ПИН-2016 |

**Москва 2019**

Основные понятия объектно-ориентированной технологии.

С чего же начинается создание объектно-ориентированной программы?

Конечно, с объектно-ориентированного анализа (ООА — object-oriented analysis), который направлен на создание моделей реальной действительности на основе объектно-ориентированного мировоззрения. Объектно-ориентированный анализ (ООА) — это методология, при которой требования к системе воспринимаются с точки зрения классов и объектов, прагматически выявленных в предметной области.

На результатах QOA формируются модели, на которых основывается объектно-ориентированное проектирование (object-oriented design, OOD).

Объектно-ориентированное проектирование (ООП) — это методология проектирования, соединяющая в себе процесс объектной декомпозиции и приемы представления логической и физической, а также статической и динамической моделей проектируемой системы.

Что же такое объектно-ориентированное программирование (ООПр) (object-oriented programming)? Программирование прежде всего подразумевает правильное и эффективное использование механизмов конкретных языков программирования. Объектно-ориентированное программирование — это процесс реализации программ, основанный на представлении программы в виде совокупности объектов. ООПр предполагает, что любая функция (процедура) в программе представляет собой метод объекта некоторого класса, причем класс должен формироваться в программе естественным образом, как только в программе возникает необходимость описания новых физических предметов или их абстрактных понятий (объектов программирования). Каждый новый шаг в разработке алгоритма также должен представлять собой разработку нового класса на основе уже существующих классов, т. е. технология ООПр иначе может быть названа как программирование "от класса к классу".

Можно ли реализовать объектно-ориентированную программу не на объектно-ориентированных языках? Ответ, скорее всего, положителен, хотя придется преодолеть ряд трудностей. Ведь главное, что требуется, — это реализовать объектную модель. Сокрытие информации при использовании обычных языков, в принципе, можно реализовать сокрытием доступности вызовов подпрограмм в файлах (Unit). Инкапсуляцию объектов можно достичь как и в объектно-ориентированных языках написанием отдельных подпрограмм. Далее можно считать, что каждый объект порождается от своего уникального класса. Конечно, иерархии классов в таком проекте не будет и для достижения параллелизма придется писать код для организации вызова к исполнению как бы сразу нескольких копий процедур, но программа при этом будет вполне объектно-ориентированной.

Основные понятия, используемые в объектно-ориентированных языках.

Класс в одном из значений этого термина обозначает тип структурированных данных.

Объект — это структурированная переменная типа класс. Каждый объект является представителем (экземпляром) определенного класса. В программе может быть несколько объектов, являющихся экземплярами одного и того же класса. Все объекты — экземпляры данного класса — аналогичны друг другу, поскольку имеют одинаковый интерфейс, один и тот же набор операций (методов) и полей, определяемых в их классе. Интерфейс класса иногда называют особенностями класса.

Класс является описанием того, как будет выглядеть и вести себя его представитель. Обычно проектируют класс как образование (матрицу), отвечающее за создание своих новых представителей (экземпляров или объектов). Экземпляр объекта создается при помощи особого метода класса, называемого конструктором, так как необходимо создать экземпляр, прежде чем он станет активным и начнет взаимодействовать с окружающим миром. Уничтожение экземпляров поддерживает сам активный экземпляр, имеющий соответствующий метод — деструктор.

*Объект*— это структурированная переменная типа класс, содержащая всю информацию о некотором физическом предмете или реализуемом в программе понятии.

*Объект —*это логическая единица, которая содержит данные и правила (методы с кодом алгоритма) (см. рис. 1.8). Другими словами, объект — это расположенные в отдельном участке памяти:

— порция данных объекта или атрибуты исходных данных, называемые еще полями, членами данных *(data members),*значения которых определяют текущее состояние объекта;

— методы объекта *(methods,*в разных языках программирования еще называют подпрограммами, действиями, *member functions*или функциями-членами), реализующие действия (выполнение алгоритмов) в ответ на их вызов в виде переданного сообщения;

— часть методов, называемых свойствами (property), которые, в свою очередь, определяют поведение объекта, т. е. его реакцию на внешние воздействия (в ряде языков программирования свойства оформляются особыми операторами).

Объекты в программах воспроизводят все оттенки явлений реального мира: "рождаются" и "умирают"; меняют свое состояние; запускают и останавливают процессы; "убивают" и "возрождают" другие объекты.

Объявления классов определяют уже описанные три характеристики объектов: поля объекта, методы объекта, свойства объектов. Также в объявлениях может указываться предок данного класса.

В соответствии с описанием класса внутри объекта данные и методы могут быть как открытыми по интерфейсу *public,*так и сокрытыми *private.*

Во время выполнения программы объекты взаимодействуют друг с другом посредством вызова методов вызываемого объекта — в этом и заключается передача сообщений. Для того чтобы объект послал сообщение другому объекту, в большинстве языков программирования требуется после указания имени вызываемого объекта записать вызов подпрограммы (метода) с соответствующим именем и указанием необходимых фактических параметров (аргументов). Получив сообщение, объект-получатель начинает выполнять код вызванной подпрограммы (метода) с полученными значениями аргументов. Таким образом, функционирование программы (выполнение всего алгоритма программы) осуществляется последовательным вызовом методов от одного объекта к другому.

Хотя можно получить прямой доступ к полям объекта, использование такого подхода не поощряется. Одно из больших преимуществ ООПр — это инкапсуляция, предназначенная для разрешения работы с данными в полях объектов только через сообщения. Для реализации методов обработки таких сообщений используются свойства. Свойства — это особым образом оформленные методы, предназначенные как для чтения и контролируемого изменения внутренних данных объекта (полей), так и выполнения действий, связанных с поведением объекта.

Так, например, если в заданном месте экрана уже отображена какая-то строка и мы хотим изменить положение строки на экране, то мы посылаем объекту новое значение свойства в виде набора нужных координат. Далее свойство автоматически трансформируется в вызов метода, который изменит значение поля координат отображения строки и выполнит действия по уничтожению изображения строки на прежнем месте экрана, а также по отображению строки в новом месте экрана.

Можно выделить несколько преимуществ инкапсуляции.

*Преимущество 1.*Надежность данных. Можно предотвратить изменение элемента данных, выполнив в свойстве (методе) дополнительную проверку значения на допустимость. Тем самым можно гарантировать надежное состояние объекта.

*Преимущество 2.*Целостность ссылок. Перед доступом к объекту, связанному с данным объектом, можно удостовериться, что косвенное поле содержит корректное значение (ссылку на экземпляр).

*Преимущество 3.*Предусмотренные побочные эффекты. Можно гарантировать, что каждый раз, когда выполняется обращение к полю объекта, синхронно с ним выполняется какое-либо специальное действие.

*Преимущество 4.*Сокрытие информации. Когда доступ к данным осуществляется только через методы, можно скрыть детали реализации объекта. Позднее, если реализация изменится, придется изменить лишь реализацию методов доступа к полям. Те же части программы, которые использовали этот класс, не будут затронуты.

Весьма удобно рассматривать объекты как попытку создания *активных данных.*Смысл, вкладываемый в слова "объект представляет собой активные данные", основан на объектно-ориентированной парадигме выполнения операций, состоящей в посылке сообщений. В посылаемых объекту сообщениях указывается, что мы хотим, что бы он выполнил. Так, например, если мы хотим вывести на экране строку, то мы посылаем объекту строки сообщение, чтобы он изобразил себя. В этом случае строка — это уже не пассивный кусок текста, а активная единица, знающая, как правильно производить над собой различные действия.

Одна из фундаментальных концепций ООП — это понятие наследования классов, устанавливающее между двумя классами отношения "родитель-потомок".

Наследование — отношение самого высокого уровня и играет важную роль на стадии проектирования. Наследование — это определение класса и затем использование его для построения иерархии производных классов, причем каждый класс-потомок наследует от класса-предка интерфейс всех классов-предков в виде доступа к коду их методов и данным. При этом, возможно, переопределение или добавление как новых данных, так и методов.

*Класс-предок —*это класс, предоставляющий свои возможности и характеристики другим классам через механизм наследования. Класс, который использует характеристики другого класса посредством наследования, называется его *классом-потомком.*

Итак, наследование проявляется в том, что любой класс-потомок имеет доступ или, другими словами, наследует практически все ресурсы (методы, поля и свойства) родительского класса и всех предков до самого верхнего уровня иерархии.

Рассмотрим, как информация, содержащаяся в классе-потомке, может переопределять информацию, наследуемую от предков. Очень часто при реализации такого подхода метод, соответствующий подклассу, имеет то же имя, что и соответствующий метод в родительском классе. При этом для поиска метода, подходящего для обработки сообщения, используется следующее правило. Поиск метода, который вызывается в ответ на определенное сообщение, начинается с методов, принадлежащих классу получателя. Если подходящий метод не найден, то поиск продолжается до родительского класса. Поиск продвигается вверх по цепочке родительских классов до тех пор, пока не будет найден нужный метод или пока не будет исчерпана последовательность родительских классов. В первом случае выполняется найденный метод, во втором выдается сообщение об ошибке. Во многих языках программирования уже на этапе компилирования, а не при выполнении программы определяется, что подходящего метода нет вообще и выдается сообщение об ошибке.

Семантически наследование описывает отношение типа *"is-a".*Например, медведь есть млекопитающее, дом есть недвижимость и "быстрая сортировка" есть сортирующий алгоритм. Таким образом, наследование порождает иерархию "обобщение — специализация", в которой подкласс представляет собой специализированный частный случай своего суперкласса. "Лакмусовая бумажка" наследования — обратная проверка: так, если *В*не есть *А,*то *В*не стоит производить от *А.*

Повторное использование — это использование в программе класса для создания экземпляров или в качестве базового для создания нового класса, наследующего часть или все характеристики родителя. Порождая классы от базовых, вы эффективно повторно используете код базового класса для собственных нужд. Повторное использование сокращает объем кода, который необходимо написать и оттестировать при реализации программы, что сокращает объемы труда.

Таким образом, наследование выполняет в ООП несколько важных функций:

• моделирует концептуальную структуру предметной области;

• экономит описания, позволяя использовать их многократно для задания разных классов;

• обеспечивает пошаговое программирование больших систем путем многократной конкретизации классов.

Ряд языков, например Object Pascal, описание которого дается в приложении 4, поддерживает модель наследования, известную как простое наследование и которая ограничивает число родителей конкретного класса одним. Другими словами, определенный пользователем класс имеет только одного родителя. Схема иерархии классов в этом случае представляет собой ряд одиночно стоящих деревьев *(hierarchical classification).*

Более мощная модель сложного наследования, называемая множественным наследованием, в которой каждый класс может, в принципе, порождаться сразу от нескольких родительских классов, наследуя поведение всех своих предков. Множественное наследование не поддерживается в Delphi, но поддерживается в Visual C++ и ряде других языков. При множественном наследовании составляется уже не схема иерархии, а сеть, которая может включать деревья со сросшимися кронами.

Обычно если объекты соответствуют конкретным сущностям реального мира, то классы являются абстракциями, выступающими в роли понятий. Между классами, как между понятиями, существует иерархическое отношение конкретизации, связывающее класс с классом-потомком. Это отношение реализуется в системах ООП механизмом наследования. Наследование — это способность одного класса использовать характеристики другого.

Наследование позволяет практически без ограничений последовательно строить и расширять классы, созданные вами или кем-то еще. Начиная с самых простых классов можно создавать производные классы по возрастающей сложности, которые не только легки при отладке, но и просты по внутренней структуре.

Множественное наследование многие аналитики считают "вредным" механизмом, приводящим к сложно разрешимым проблемам проектирования и реализации. В языках с отсутствующим множественным наследованием целей множественного наследования достигают *агрегированием объектов с дополнительным делегированием полномочий.*

На агрегировании основана работа таких систем визуального программирования, как Delphi, C++ Builder. В этих системах имеется порождающий объект пользователя класс-форма (пустое окно Windows). Системы обеспечивают подключение к форме через указатели нужных пользователю объектов, например кнопок, окон редакторов и т. д. При перерисовке формы на экране монитора как бы одновременно с ней перерисовываются изображения агрегированных объектов. Более того, при активизации формы агрегированные объекты также становятся активными: кнопки начинают нажиматься, а в окна редакторов можно начинать вводить информацию.

Одним из базовых понятий технологии ООП является полиморфизм. Термин "полиморфизм" имеет греческое происхождение и означает приблизительно "много форм" *(poly —*много, *morphos —*форма).

*Полиморфизм —*это средство для придания различных значений одному и тому же событию в зависимости от типа обрабатываемых данных, т. е. полиморфизм определяет различные формы реализации одноименного действия (см. рис. 8.2.).

Целью полиморфизма применительно к объектно-ориентированному программированию является использование одного имени для задания общих для класса действий, причем каждый объект имеет возможность по-своему реализовать это действие своим собственным, подходящим для него кодом.

Полиморфизм является предпосылкой для расширяемости объектно-ориентированных программ, поскольку он предоставляет способ старым программам воспринимать новые типы данных, которые не были определены во время написания программы.

Противоположность полиморфизму называется мономорфизмом; он характерен для языков с сильной типизацией и статическим связыванием (Ada).

В более общей трактовке полиморфизм — это способность объектов, принадлежащих к разным типам, демонстрировать одинаковое поведение; способность объектов, принадлежащих к одному типу, демонстрировать разное поведение.

Рассмотрим "вырожденный пример" полиморфизма. В MS DOS есть понятие "номер прерывания", за которым скрывается адрес в памяти. Поместите в ту же ячейку другой адрес — и программы начнут вызывать процедуру с другим "именем" и из другого модуля. Как видно из примера, принцип полиморфизма можно реализовать и не в объектно-ориентированных программах.

Ряд авторов книг по теории объектно-ориентированного проектирования соотносят термин "полиморфизм" с разными понятиями, например понятием перегрузки; для обозначения одного-двух или большего количества механизмов полиморфизма; чистого полиморфизма.

*Перегрузка функций.*Одним из применений полиморфизма в C++ является перегрузка функций. Она дает одному и тому же имени функции различные значения. Например, выражение *а*+ *b*имеет различные значения, в зависимости от типов переменных *а*и *b*(допустим, если это числа, то "+" означает сложение, а если строки, — то склейку этих строк или вообще сложение комплексных чисел, если *а и b*комплексного типа). Перегрузка оператора "+" для типов, определяемых пользователем, позволяет использовать их в большинстве случаев так же, как и встроенные типы. Двум или более функциям (операция — это тоже функция) может быть дано одно и то же имя. Но при этом функции должны отличаться сигнатурой (либо типами параметров, либо их числом).

Полиморфный метод в C++ называется *виртуальной функцией,*позволяющей получать ответы на сообщения, адресованные объектам, точный вид которых неизвестен. Такая возможность является результатом позднего связывания. При позднем связывании адреса определяются динамически во время выполнения программы, а не статически во время компиляции как в традиционных компилируемых языках, в которых применяется *раннее связывание.*Сам процесс связывания заключается в замене виртуальных функций на адреса памяти.

Виртуальные методы определяются в родительском классе, а в производных классах происходит их доопределение и для них создаются новые реализации. Основой виртуальных методов и динамического полиморфизма являются указатели на производные классы. При работе с виртуальными методами сообщения передаются как указатели, которые указывают на объект вместо прямой передачи объекту.

Практический смысл полиморфизма заключается в том, что он позволяет посылать общее сообщение о сборе данных любому классу, причем и родительский класс, и классы-потомки ответят на сообщение соответствующим образом, поскольку производные классы содержат дополнительную информацию. Программист может сделать регулярным процесс обработки несовместимых объектов различных типов при наличии у них такого полиморфного метода.

Этапы и модели объектно-ориентированной технологии.

Почему в начале процесса проектирования работу начинают с анализа функционирования или поведения системы? Дело в том, что поведение системы обычно известно задолго до остальных ее свойств. Программа должна выполнять набор действий, согласно выявленным ее функциям. Процесс разработки модели в форме функциональной спецификации уже был изложен ранее в гл. 5.

Объектно-ориентированная технология создания программ основывается на так называемом объектном подходе. Одним из проявлений этого подхода является то, что сначала довольно долго создаются и оптимизируются объектная модель и иные модели и лишь затем осуществляется кодирование.

Обычно проектируемая программная система первоначально представляется в виде трех взаимосвязанных моделей:

1) объектной модели, которая представляет статические, структурные аспекты системы;

2) динамической модели, которая описывает работу отдельных частей системы;

3) функциональной модели, в которой рассматривается взаимодействие отдельных частей системы (как по данным, так и по управлению) в процессе ее работы.

Эти три вида моделей должны позволить рассматривать три взаимно-ортогональных представления системы в одной системе обозначений.

Объектная модель на более поздних этапах проектирования дополняется моделями, отражающими как логическую (классы и объекты), так и физическую структуру системы (процессы и деление на компоненты, файлы или модули).

Поскольку при разработке объектно-ориентированного проекта используется множество моделей, которые необходимо увязать в единое целое, далее в гл. 10 рассматриваются средства автоматизации составления, верификации (проверки) и графической визуализации этих моделей.

Процесс построения ***объектной модели***включает в себя следующие, возможно, повторяющиеся до достижения приемлемого качества модели этапы:

1) определение объектов;

2) подготовку словаря объектов с целью исключения схожих (синонимичных) понятий и уточнения имен, классификацию объектов, выделение классов;

3) определение взаимосвязей между объектами;

4) определение атрибутов объектов и методов (определение уровней доступа и проектирование интерфейсов классов);

5) исследование качества модели.

Теперь, используя функциональную модель, можно начинать работу с динамической моделью, наделяя объекты необходимыми методами и данными.

Модели, разработанные на первой фазе жизненного цикла системы, продолжают использоваться на всех последующих его фазах, облегчая программирование системы, ее отладку и тестирование, сопровождение и дальнейшую модификацию.

Объектная модель описывает структуру объектов, составляющих систему, их атрибуты, операции, взаимосвязи с другими объектами. В объектной модели должны быть отражены те понятия и объекты реального мира, которые важны для разрабатываемой системы. В объектной модели отражается прежде всего прагматика разрабатываемой системы, что выражается в использовании терминологии прикладной области, связанной с использованием разрабатываемой системы.

Прагматика определяется целью разработки программной системы: для обслуживания покупателей железнодорожных билетов, управления работой аэропорта, обслуживания чемпионата мира по футболу и т. п. В формулировке цели участвуют предметы и понятия реального мира, имеющие отношение к разрабатываемой программной системе.

Объектную модель можно описать следующим образом:

1) основные элементы модели — объекты и сообщения;

2) объекты создаются, используются и уничтожаются подобно динамическим переменным в обычных языках программирования;

3) выполнение программы заключается в создании объектов и передаче им последовательности сообщений.

Объектная модель базируется на четырех главных принципах: абстрагировании; инкапсуляции; модульности; иерархии.

Эти принципы являются *главными*в том смысле, что без любого из них модель не будет по-настоящему объектно-ориентированной.

*Абстрагирование*концентрирует внимание на внешних особенностях объекта и позволяет отделить самые существенные особенности поведения от несущественных. Выбор правильного набора абстракций для заданной предметной области представляет собой главную задачу объектно-ориентированного проектирования.

Все абстракции обладают как статическими, так и динамическими свойствами. Например, файл как объект требует определенного объема памяти на конкретном устройстве, имеет имя и содержание. Эти атрибуты являются статическими свойствами. Конкретные же значения каждого из перечисленных свойств динамичны и изменяются в процессе использования объекта: файл можно увеличить или уменьшить, изменить его имя и содержимое.

Абстракция и инкапсуляция дополняют друг друга: абстрагирование направлено на наблюдаемое поведение объекта, а инкапсуляция занимается внутренним устройством. Чаще всего инкапсуляция дополняется сокрытием информации, т. е. маскировкой всех внутренних деталей, не влияющих на внешнее поведение. Объектный подход предполагает, что собственные ресурсы, которыми могут манипулировать только методы самого объекта, скрыты от внешних компонент.

При объектно-ориентированном проектировании необходимо физически разделить классы и объекты, составляющие логическую структуру проекта. Такое разделение делает возможным повторно использовать во все новых проектах код модулей, написанных ранее. Модулю в данном контексте соответствует отдельный файл исходного текста. На выбор разбиения на модули могут влиять и некоторые внешние обстоятельства. При коллективной разработке программ распределение работы осуществляется, как правило, по модульному принципу, и правильное разделение проекта минимизирует связи между участниками.

Таким образом, принципы абстрагирования, инкапсуляции и модульности являются взаимодополняющими. Объект логически определяет границы определенной абстракции, а инкапсуляция и модульность делают их физически незыблемыми.

Имея выявленные объекты, можно приступить к выявлению классов. Классы чаще всего строятся постепенно, начиная от простых родительских классов и заканчивая более сложными. Непрерывность процесса основана на наследовании. Каждый раз, когда из предыдущего класса производится последующий, производный класс наследует какие-то или все родительские качества, добавляя к ним новые. Завершенный проект может включать десятки и сотни классов, но часто все они произведены от считанного количества родительских классов.

Технология визуального программирования.

Н ачальные шаги технологии визуального программирования определяются оболочкой самой среды визуального программирования. Сначала создаются экранные формы простейшей буксировкой мыши. В инспекторе объектов производится настройка их свойств путем заполнения отдельных полей. На главную форму помимо визуальных компонент наносятся невизуальные компоненты. Формы объединяются в единый проект. Далее в соответствии со сценарием диалога программируются методы события основной и подчиненных форм. Программы "пустых" методов событий появляются в окне редактора после нажатия соответствующих клавиш или действий мыши. "Пустые" методы дополняются определенными операторами активации и дезактивации форм. По окончании начальных шагов получается работающий "скелет" программы с источниками данных из файловых баз данных и со сгенерированными формами документов, выводимых на печать. Исследователь (Browser) обеспечивает визуализацию схемы иерархии классов полученного "скелета" программы. Другими словами, технический проект реализованной части программы формируется автоматически.

Дальнейшая разработка программы ведется по технологии объектно-ориентированного программирования. Можно часть программы реализовать по технологии структурного программирования. Некоторые недостающие визуальные и невизуальные компоненты получаются модификацией исходных текстов наиболее близких прототипов имеющихся компонент. Рекомендуется новые компоненты помещать в палитру компонент. Это облегчит их повторное использование в данной или последующих разработках. Код, относящийся только к данной разработке, набирается по тексту программы.

ВЫВОДЫ

• Визуальное программирование во многом автоматизирует труд программиста по написанию программ.

• Визуальное программирование — одна из самых популярных парадигм программирования на данный момент. Оно базируется на технологии ООП.

• Среда визуального программирования поддерживает работу браузеров (Browser), при помощи которых можно автоматически получить документацию по структуре программы.

• Основным элементом в средствах визуального программирования является компонент. Компоненты бывают визуальными и невизуальными.

• Технология визуального программирования состоит в следующем: создание экранных форм, нанесение визуальных и невизуальных компонент, программирование событий и методов оконных форм.

CASE-средства и визуальное моделирование.

Тенденции развития современных информационных технологий приводят к постоянному усложнению автоматизированных систем (АС). Для борьбы со сложностью проектов в настоящее время созданы системы автоматизированного проектирования (САПР) самих программных проектов.

Для успешной реализации проекта объект проектирования (АС) должен быть прежде всего адекватно описан, должны быть построены полные, а также непротиворечивые функциональные и информационные модели АС. Накопленный к настоящему времени опыт проектирования АС показывает, что это трудоемкая и длительная по времени работа.

Все это способствовало появлению программно-технологических средств особого назначения — CASE-средств, реализующих CASE-инженерию создания и сопровождения АС. Термин "CASE" (Computer Aided Software Engineering) используется в настоящее время в весьма широком смысле. Первоначальное значение термина CASE, ограниченное вопросами автоматизации разработки только лишь программного обеспечения (ПО), в настоящее время приобрело новый смысл, охватывающий процесс разработки сложных АС в целом.

Теперь под термином *"CASE-средства"*понимаются программные средства, поддерживающие процессы создания и сопровождения АС, включая анализ и формулировку требований, проектирование прикладного ПО (приложений) и баз данных, генерацию кода, тестирование, документирование, обеспечение качества, конфигурационное управление и управление проектом, а также другие процессы. CASE-средства вместе с системным ПО и техническими средствами образуют среду разработки АС.

**CASE-технология**представляет собой методологию проектирования АС, а также набор инструментальных средств, позволяющих в наглядной форме моделировать предметную область, анализировать эту модель на всех этапах разработки и сопровождения АС и разрабатывать приложения в соответствии с информационными потребностями пользователей. Большинство существующих CASE-средств основано на методологиях структурного (в основном) или объектно-ориентированного анализа и проектирования, использующих спецификации в виде диаграмм или текстов для описания внешних требований, связей между моделями системы, динамики поведения системы и архитектуры программных средств.

На основании анкетирования более тысячи американских фирм фирмой "Systems Development Inc." в 1996 г. был составлен обзор передовых технологий (Survey of Advanced Technology). Согласно этому обзору CASE-технология в настоящее время попала в разряд наиболее стабильных информационных технологий (ее использовала половина всех опрошенных пользователей более чем в трети своих проектов, из них 85 % завершились успешно). Однако несмотря на все потенциальные возможности CASE-средств, существует множество примеров их неудачного внедрения. В связи с этим необходимо отметить следующее:

— CASE-средства не обязательно дают немедленный эффект, он может быть получен только спустя какое-то время;

— реальные затраты на внедрение CASE-средств обычно намного превышают затраты на их приобретение;

— CASE-средства обеспечивают возможности для получения существенной выгоды только после успешного завершения процесса их внедрения.

Для успешного внедрения CASE-средств организация должна обладать такими качествами, как:

• *технология*— понимание ограниченности существующих возможностей и способность принять новую технологию;

• *культура —*готовность к внедрению новых процессов и взаимоотношений между разработчиками и пользователями;

• *управление —*четкое руководство и организованность по отношению к наиболее важным этапам и процессам внедрения.

Изучая требования к системе, вы берете за основу запросы пользователей и далее преобразуете их в такую форму, которую ваша команда сможет понять и реализовать. На основе этих требований вы генерируете код. Формально преобразуя требования в код, вы гарантируете их соответствие друг другу, а также возможность в любой момент вернуться от кода к породившим его требованиям. Этот процесс называется *моделированием.*Моделирование позволяет проследить путь от запросов пользователей к требованиям модели и затем к коду и обратно, не теряя при этом своих наработок.

**Визуальным моделированием**называют процесс графического представления модели с помощью некоторого стандартного набора графических элементов. Наличие стандарта жизненно необходимо для реализации одного из преимуществ визуального моделирования — *коммуникации.*Общение между пользователями, разработчиками, аналитиками, тестировщиками, менеджерами и всеми остальными участниками проекта является основной целью графического визуального моделирования.

Созданные модели представляются всем заинтересованным сторонам, которые могут извлечь из них ценную информацию. Например, глядя на модель, пользователи визуализируют свое взаимодействие с системой. Аналитики увидят взаимодействие между объектами модели. Разработчики поймут, какие объекты нужно создать и что эти объекты должны делать. Тестировщики визуализируют взаимодействие между объектами, что позволит им построить тесты. Менеджеры увидят как всю систему в целом, так и взаимодействие ее частей. Наконец, руководители информационной службы, глядя на высокоуровневые модели, поймут, как взаимодействуют друг с другом системы в их организации. Таким образом, визуальные модели предоставляют мощный инструмент, позволяющий показать разрабатываемую систему всем заинтересованным сторонам.