Лекция 2. Программные продукты как сложные системы

Целью инженерной деятельности является создание качественных продуктов и систем, выработка качественных решений. Целью программирования, как одного из видов инженерной деятельности, является создание качественных программ.

Наиболее важными характеристиками качества программных средств (ПС) являются[I]:

- корректность;
- надёжность, устойчивость к сбоям, работоспособность;
- расширяемость (модифицируемость), повторная используемость.

Для достижения указанных характеристик качества разработаны и практически апробированы различные концепции, стили программирования. Можно сказать, что целью всех концепций программирования является повышение качества программных систем.

Достижению качества программных средств препятствует *сложность*, которая лежит в природе компьютерных программ. Разработка программ требует анализа сложного неорганизованного реального мира, точного и полного определения зависимостей и исключений, проектирования абсолютно верных решений. Все остальные технические цели разработки программного обеспечения (ПО) вторичны по отношению к управлению сложностью[II].

В толковом словаре сказано, что сложность означает «иметь сложное качество или состояние», т.е. иметь много различных взаимосвязанных частей, структур или элементов и, следовательно, быть трудно понимаемым полностью», или «включать множество частей, аспектов, деталей, понятий, требующих для понимания или овладения серьёзного исследования или рассмотрения».

Примеры сложных систем

Рассмотрим примеры сложных систем в различных предметных областях[III].

Структура персонального компьютера. Персональный компьютер (ПК) — прибор умеренной сложности. Большинство ПК состоит из одних и тех же основных элементов: системной платы, монитора, клавиатуры и устройства внешней памяти какого-либо типа. Мы можем взять любую из этих частей и разложить её в свою очередь на составляющие. Системная плата, например, содержит оперативную память, центральный процессор (ЦП) и шину, к которой подключены периферийные устройства. Каждую из этих частей можно также разложить на составляющие: ЦП состоит из регистров и схем управления, которые сами состоят из ещё более простых деталей: диодов, транзисторов и т.д.

Это пример сложной иерархической системы. Персональный компьютер нормально работает благодаря чёткому совместному функционированию всех его составных частей. Вместе эти части образуют логическое целое. Мы можем понять, как работает компьютер, только потому, что можем рассматривать отдельно каждую его составляющую. Таким образом, можно изучать устройства монитора и жёсткого диска независимо друг от друга. Аналогично можно изучать арифметическую часть ЦП, не рассматривая при этом подсистему памяти.

Уровни этой иерархии представляют различные уровни абстракции, причём один надстроен над другим и каждый может быть рассмотрен отдельно. На каждом уровне абстракции мы находим набор устройств, которые совместно обеспечивают некоторые функции более высокого уровня, и выбираем уровень абстракции, исходя из наших специфических потребностей.

Структура растений. Растения — это сложные многоклеточные организмы. В результате совместной деятельности различных органов растений происходят такие сложные типы поведения, как фотосинтез.

Растение состоит из трёх основных частей: корни, стебли и листья. Каждая из них имеет свою особую структуру. Корень, например, состоит из корневых отростков, корневых волосков, верхушки корня и т.д. Рассматривая срез листа, мы видим его эпидермис, мезофилл и сосудистую ткань. Каждая из этих структур, в свою очередь, представляет собой набор клеток. Внутри каждой клетки можно выделить следующий уровень, который включает хлоропласт, ядро и т.д. Так же, как у компьютера, части растения образуют иерархию, каждый уровень которой обладает собственной независимой сложностью.

[I] Мейер Б. Объектно-ориентированное конструирование программных систем – М.: Русская Редакция, 2005.

[II] Макконнелл С. Совершенный код – СПб.: Питер, 2005 – 896 с.

[III] Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений – М.:Вильямс, 2008 – 720 с

Все части на одном уровне абстракции взаимодействуют вполне определённым образом. Например, на высшем уровне абстракции, корни отвечают за поглощение из почвы воды и минеральных веществ. Корни взаимодействуют со стеблями, которые передают эти вещества листьям. Листья в свою очередь используют воду и минеральные вещества, доставляемые стеблями, и производят при помощи фотосинтеза необходимые элементы.

Для каждого уровня абстракции всегда чётко разграничено «внешнее» и «внутреннее». Например, можно установить, что части листа совместно обеспечивают функционирование листа в целом и очень слабо взаимодействуют или вообще прямо не взаимодействуют с элементами корней. Проще говоря, существует чёткое разделение функций различных уровней абстракции.

В компьютере транзисторы используются как в схеме ЦП, так и жёсткого диска. Аналогично этому большое число «унифицированных элементов» имеется во всех частях растения. Так Создатель достигал экономии средств выражения. Например, клетки служат основными строительными блоками всех структур растения; корни, стебли и листья растения состоят из клеток. И хотя любой из этих исходных элементов действительно является клеткой, существует огромное количество разнообразных клеток. Есть клетки, содержащие и не содержащие хлоропласт, клетки с оболочкой, проницаемой и непроницаемой для воды, и даже живые и умершие клетки.

Структура социальных институтов. Как последний пример сложных систем рассмотрим структуру общественных институтов. Люди объединяются в группы для решения задач, которые не могут быть решены индивидуально. Одни организации быстро распадаются, другие функционируют на протяжении нескольких поколений. Чем больше организация, тем отчётливее проявляется в ней иерархическая структура. Транснациональные корпорации состоят из компаний, которые, в свою очередь, состоят из отделений, содержащих различные филиалы. Последним принадлежат уже отдельные офисы и т.д. Границы между частями организации могут изменяться, и с течением времени может возникнуть новая, более стабильная иерархия.

Признаки сложной системы

Любая сложная система имеет пять признаков.

1. Сложные системы часто являются *иерархическими* и состоят из взаимозависимых *подсистем*, которые в свою очередь также могут быть разделены на подсистемы, и т.д., вплоть до самого низкого уровням. Архитектура сложных систем складывается и из компонентов, и из иерархических отношений этих компонентов.
2. Выбор, какие компоненты в данной системе считаются элементарными, относительно произволен и в большой степени оставляется на усмотрение исследователя. Низший уровень для одного наблюдателя может оказаться достаточно высоким для другого.
3. Внутрикомпонентная связь обычно сильнее, чем связь между компонентами. Это различие внутрикомпонентных и межкомпонентных взаимодействий обуславливает разделение функций между частями системы и дает возможность относительно изолированно изучать каждую часть.
4. Иерархические системы обычно состоят из немногих типов подсистем, по-разному скомбинированных и организованных. Иными словами, разные сложные системы содержат одинаковые структурные части. Эти части могут использовать общие более мелкие компоненты.
5. Любая работающая сложная система является результатом развития работавшей более простой системы... Сложная система, спроектированная «с нуля», никогда не заработает. Следует начинать с работающей простой системы.

В процессе развития системы объекты, первоначально рассматривавшиеся как сложные, становятся элементарными, и из них строятся более сложные системы. Более того, невозможно сразу правильно создать элементарные объекты: с ними надо сначала поработать, чтобы больше узнать о реальном поведении системы, и затем уже совершенствовать их.

Опыт показывает, что наиболее успешны те программные системы, в которых заложены хорошо продуманные структуры классов и объектов и которые обладают пятью признаками сложных систем, описанными выше.

Когда мы начинаем анализировать сложную программную систему, в ней обнаруживается много составных частей, которые взаимодействуют друг с другом различными способами, причем ни сами части системы, ни способы их взаимодействия не обнаруживают никакого сходства. Это пример неорганизованной сложности.

Эксперименты психологов показывают, что максимальное количество структурных единиц информации, за которыми человеческий мозг может одновременно следить, приблизительно равно семи плюс-минус два. Вероятно, это связано с объемом краткосрочной памяти у человека. Дополнительным ограничивающим фактором является скорость обработки мозгом поступающей информации: на восприятие каждой новой единицы информации ему требуется около 5 секунд.

Таким образом, мы оказались перед серьезной дилеммой. Сложность программных систем возрастает, но способность нашего мозга справиться с этой сложностью ограничена. Как же нам выйти из создававшегося затруднительного положения?

Рассмотрение любой сложной системы требует применения техники **декомпозиции** — разбиения на составляющие элементы. При проектировании сложной программной системы необходимо разделять её на всё меньшие и меньшие подсистемы, каждую из которых можно совершенствовать независимо. В этом случае мы не превысим пропускной способности человеческого мозга: для понимания любого уровня системы нам необходимо одновременно держать в уме информацию лишь о немногих её частях.

Известны две схемы декомпозиции: *алгоритмическая декомпозиция* и *объектно-ориентированная декомпозиция*.

В основе алгоритмической декомпозиции лежит разбиение по действиям — алгоритмам. Большинство из нас формально обучено структурному проектированию «сверху вниз», и мы воспринимаем декомпозицию как обычное разделение алгоритмов, где каждый модуль системы выполняет один из этапов общего процесса.

Объектно-ориентированная декомпозиция обеспечивает разбиение по автономным лицам — объектам реального (или виртуального) мира. Эти объекты — более «крупные» элементы, каждый из них несёт в себе и описания действий, и описания данных.

Разделение по алгоритмам концентрирует внимание на порядке происходящих событий, а разделение по объектам придаёт особое значение агентам, которые являются либо объектами, либо субъектами действия.

Объектная декомпозиция имеет несколько чрезвычайно важных преимуществ перед алгоритмической. Объектная декомпозиция уменьшает размер программных систем за счёт повторного использования общих механизмов, что приводит к существенной экономии выразительных средств. Объектно-ориентированные системы более гибки и проще эволюционируют со временем, потому что их схемы базируется на устойчивых промежуточных формах. Действительно, объектная декомпозиция существенно снижает риск при создании сложной программной системы, так как она развивается из меньших систем, в которых мы уже уверены. Более того, объектная декомпозиция помогает нам разобраться в сложной программной системе, предлагая нам разумные решения относительно выбора подпространства большого пространства состояний.

Объектная декомпозиция

Объектно-ориентированная технология основывается на так называемой *объектной модели*. Объектно-ориентированная технология включает в себя *объектно-ориентированное программирование*, *проектирование* и *анализ*.

Объектно-ориентированное программирование — это методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования.

В данном определении можно выделить три части:

1) ООП использует в качестве базовых элементов *объекты,* а не алгоритмы;
2) каждый объект является *экземпляром* какого-либо определенного *класса*;
3) классы организованы *иерархически*.

Программа будет объектно-ориентированной только при соблюдении всех трёх указанных требований. В частности, программирование, не основанное на иерархических отношениях, не относится к ООП, а называется *программированием на основе абстрактных типов данных*.

При использовании технологии ООП решение задачи представляется в виде *результата взаимодействия отдельных функциональных элементов* некоторой системы, имитирующей процессы, происходящие в предметной области поставленной задачи.

В такой системе каждый функциональный элемент, получив некоторое входное воздействие (которое называют **сообщением**) в процессе решения задачи, выполняет заранее определённые действия (например, может изменить собственное состояние, выполнить некоторые вычисления, нарисовать окно или график и в свою очередь воздействовать на другие элементы). Процессом решения задачи управляет *последовательность сообщений*. Передавая эти сообщения от элемента к элементу, система выполняет необходимые действия.

Функциональные элементы системы, параметры и поведение которой определяются условием задачи, обладающие самостоятельным поведением (то есть «умеющие» выполнять некоторые действия, зависящие от полученных сообщений и состояния элемента) получили название *объектов*.

Процесс представления предметной области задачи в виде совокупности объектов, обменивающихся сообщениями, называется *объектной декомпозицией*.

Для того чтобы понять, о каких объектах и сообщениях идет речь при выполнении объектной декомпозиции в каждом конкретном случае, следует вспомнить, что первоначально объектный подход был предложен для разработки имитационных моделей поведения сложных систем. Набор объектов таких систем обычно определяется при анализе моделируемых процессов.

Рассмотрим пример объектной декомпозиции имитационной модели бензоколонки.

Пусть нас интересует зависимость длины очереди к бензоколонке от количества заправочных мест, параметров обслуживания каждого заправочного места и интенсивности поступления заявок на заправку топливом (рассматриваем топливо одного типа).

Задачи такого вида обычно решаются с использованием имитационных моделей. Модель программно имитирует реальный процесс с заданными параметрами, параллельно фиксируя его характеристики. Многократно повторяя процесс имитации с различными значениями параметров обслуживания или поступления заявок, исследователь получает конкретные значения характеристик, по которым строятся графики анализируемых зависимостей.

Процесс работы бензоколонки с тремя заправочными местами можно представить в виде диаграммы (рис. 2.1).

Рис. 2.1. Диаграмма обслуживания автомашин на бензоколонке

Здесь *t1*, *t2*, *t3*, … — моменты времени, когда подъезжает очередная автомашина, *Колонка 1*, *Колонка 2* и *Колонка 3* — заправочные места. Прямоугольник на диаграмме соответствует времени заправки автомашины (на диаграммах колонок) или времени ожидания в очереди (на диаграмме очереди). Номера машин при этом указаны внутри прямоугольников.

Первая подъехавшая автомашина занимает первую колонку, вторая — вторую, а третья — третью. К моменту появления четвёртой автомашины освобождается первая колонка, и четвёртая автомашина её занимает. В момент появления пятой автомашины все колонки заняты, она становится в очередь, ожидая освобождения колонки. Таким образом, автомашина, приехавшая на заправку в зависимости от наличия свободных колонок

либо сразу подъезжает к колонке, либо ожидает в очереди освобождения колонки. Имитация данного процесса может выполняться следующим образом.

Процесс поступления автомашин на заправку будет имитироваться с помощью генератора заявок на обслуживание. Обычно в качестве генератора заявок используется датчик случайных чисел, работающий по заданному закону распределения. Фактически датчик случайных чисел будет определять интервал между приходами автомашин, используя который мы сможем определить время поступления следующей автомашины.

Процесс обслуживания будет имитироваться также с помощью датчика случайных чисел, который определяет время обслуживания каждой машины в момент занятия ею колонки. Фактически, таким образом, он определяет время освобождения колонки.

В цикле имитации модель бензоколонки будет опрашивать модель потока машин и блок колонок, какое событие произойдет раньше: придет следующая автомашина или освободится колонка. Определив время и тип будущего события, модель увеличит модельное время до момента наступления ближайшего события и инициирует его обработку, передав управление соответственно генератору потока машин или блоку бензоколонок.

Получив управление, модель генератора потока машин запросит у модели блока колонок, есть ли свободные колонки. Если незанятые колонки есть, то модель генератора потока машин сформулирует запрос на занятие одной из них, а если нет, то передаст модели очереди сообщение о постановке автомашины в очередь.

Обрабатывая событие освобождения колонки, модель блока колонок запросит у модели очереди информацию о наличии машин в очереди. Если машины в очереди есть, то модель блока колонок «заберёт» одну машину из очереди и вновь займет колонку. Если машин в очереди нет, то она зафиксирует наличие свободной колонки.

Модель очереди, получая управление, будет отслеживать изменение размера очереди во времени. На основании этих данных при завершении моделирования можно определить среднюю длину очереди.

На рис. 2.2 представлена диаграмма объектов имитационной модели бензоколонки. На этой диаграмме показаны объекты и сообщения, которые эти объекты передают друг другу.

Рис. 2.2. Диаграмма объектов имитационной модели бензоколонки

Полученная модель может быть реализована в виде объектно-ориентированной программы. При программировании модели блока колонок, очереди и генератора потока автомашин будут представлены в виде объектов некоторых специально разработанных классов, а управляющая модель — в виде основной программы, инициирующей процесс моделирования. Передача сообщений в системе будет имитироваться как вызовы соответствующих методов объектов.

Объектная декомпозиция, так же как и процедурная, может применяться многократно, или быть многоуровневой. Это значит, что каждый объект может рассматриваться как система, которая состоит из элементов, взаимодействующих друг с другом через передачу сообщений. При многоуровневой декомпозиции на каждом уровне мы получаем объекты с более простым поведением, что позволяет разрабатывать системы повышенной сложности по частям.

Покажем, как происходит многоуровневая декомпозиция на том же примере. Для этого выполним декомпозицию объекта *Блок колонок*.

Модель блока колонок должна включать модели колонок и некоторый управляющий объект, который назовем *Монитором*. Монитор, получив сообщение, интерпретирует его и при необходимости генерирует сообщения моделям колонок. Например, получив сообщение-запрос о времени освобождения колонки, монитор отправит сообщения-запросы колонкам и выберет минимальное время, из сообщенных колонками. Это минимальное время он вернет модели в качестве ответа на её запрос. Получив сообщение о наступлении времени освобождения колонки, Монитор отправит соответствующие сообщения модели очереди и освобождаемой колонке (рис. 2.3).

Рис. 2.3. Объектная декомпозиция блока колонок

Безусловно, это не единственный вариант декомпозиции объекта *Блок колонок*. При желании можно найти другие механизмы реализации того же поведения. Выбор конкретного варианта в каждом случае осуществляется разработчиком.

Таким образом, становится понятно, что в процессе объектной декомпозиции имитационных моделей выделяются части предметной области, которые будут моделироваться как единое целое, обладающее собственным состоянием и поведением, и определяется характер взаимодействия этих частей.

Несколько сложнее дело обстоит с распространением идеи объектной декомпозиции на классы задач, напрямую не связанных с имитацией. По сути, в этом случае мы как бы программно *имитируем поведение* разрабатываемой системы.

Проиллюстрируем сначала на очень простом примере идею объектной декомпозиции для задач, не связанных с имитацией. Выполним объектную декомпозицию программы, которая по запросу пользователя рисует одну из двух фигур: квадрат или круг. При желании пользователь должен иметь возможность изменить цвет контура, размер фигуры и координаты её центра.

По правилам выполнения объектной декомпозиции разрабатывается имитационная модель программы. Для этого придётся проанализировать все происходящие в имитируемой системе процессы и выделить элементы, обладающие собственным поведением, воздействующие на другие элементы и/или являющиеся объектами такого воздействия.

Основной процесс системы — процесс управления рисованием фигур, указанных пользователем. Все команды пользователя должны интерпретироваться, и в результате интерпретации должны формироваться команды на рисование или изменение параметров фигур. Эти процессы можно моделировать, используя три объекта: *Монитор* (блок управления, который получает и интерпретирует команды пользователя) и два объекта *Фигура* (рис. 2.4), каждый со своими параметрами.

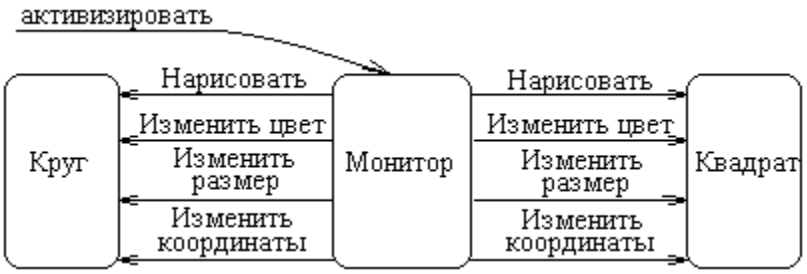

Рис. 2.4. Диаграмма объектов графического редактора

Фигуры получают следующие сообщения: «Нарисовать», «Изменить цвет контура», «Изменить размер», «Изменить координаты». Все эти сообщения инициируются *Монитором* в соответствии с командой пользователя. Получив от пользователя команду «Завершить», *Монитор* прекращает выполнение программы.

Рассмотрим более сложный пример. Попробуем выполнить объектную декомпозицию для программы «Записная книжка». Работу с программами такого типа пользователю удобнее осуществлять через меню, поэтому при запуске программы на экране должно появиться меню, содержащее набор возможных операций. В зависимости от выбора пользователя далее будут активизироваться части программы, ответственные за открытие «книжки», добавление записей или за поиск записей (рис. 2.5).

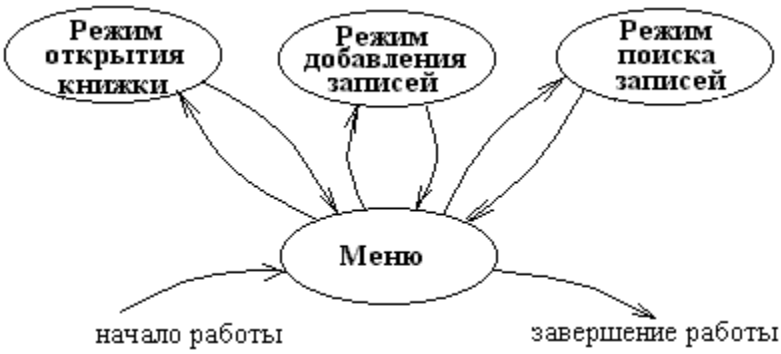

Рис. 2.5. Диаграмма состояний интерфейса пользователя (первый вариант)

На этом этапе анализа мы уже можем выделить четыре самостоятельных части программы, которые взаимодействуют в процессе ее выполнения: *Меню*, *Открытие книжки*, *Ввод записей*, *Поиск записей*. Часть *Меню* является управляющей и активизирует в процессе работы с программой остальные части для выполнения требуемых операций.

Добавим к системе ещё несколько видимых объектов-сообщений пользователю, которые будут появляться на экране при обнаружении несоответствий в процессе работы, например, «Записная книжка не найдена» активизируется *Открытием книжки* или «Нет информации об абоненте» активизируется *Поиском записей* (рис. 2.6).

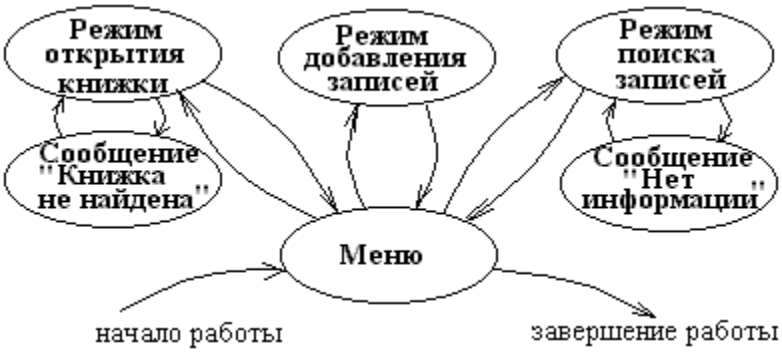

Рис. 2.6. Полная диаграмма состояний интерфейса пользователя

Объекты *Меню*, *Открытие книжки*, *Ввод записей*, *Поиск записей* в процессе работы должны получать информацию от пользователя и сообщать ему результаты работы, и соответственно должны иметь некоторое

экранное представление. Совокупность таких экранных представлений (форм) образует интерфейс с пользователем.

Помимо объектов интерфейса система содержит ещё, по крайней мере, один объект — *Файл записей*, который используется для хранения введённой информации. Этот объект должен получать от *Открытия*, *Ввода* и *Поиска* сообщения-команды, соответственно, открытия файла, добавления информации и поиска. Команда открытия должна сопровождаться именем файла, команда добавления информации — идентификационным заголовком и телефоном, а команда поиска — идентификационным заголовком.

Окончательный вариант объектной декомпозиции проектируемой системы представлен на рис. 2.7.

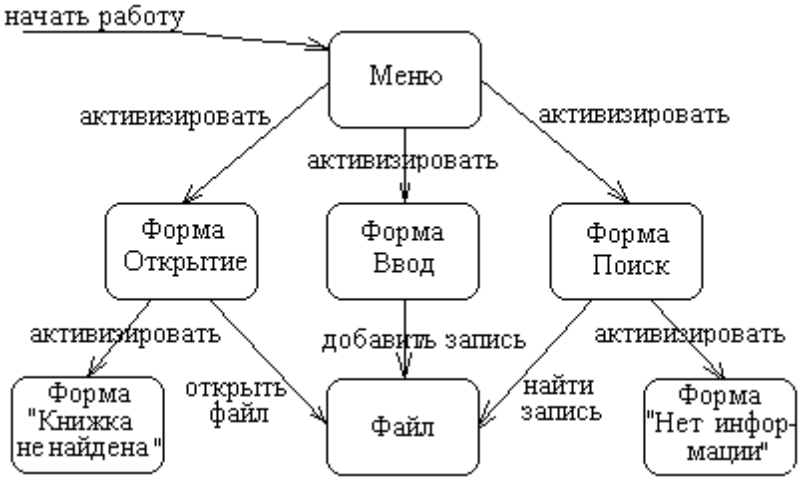

Рис. 2.7. Диаграмма объектов системы «Записная книжка»

Таким образом, можно сформулировать следующие рекомендации по выполнению объектной декомпозиции:

1. Для сложных систем объектная декомпозиция должна выполняться поэтапно: на первом этапе — объектная декомпозиция всей системы, на последующих — декомпозиция объектов, как подсистем.
2. При декомпозиции системы в целом в качестве объектов могут выделяться элементы двух типов:
   1) элементы интерфейса пользователя (окна меню, окна сообщений, окна форм ввода-вывода и т.д.);
   2) средства хранения, организации и преобразования данных (базы данных, файлы, протоколы, структуры данных и т.д.).

   При этом для каждого объекта должны определяться множество получаемых и передаваемых сообщений и основные характеристики.
3. Процесс декомпозиции прекращается при получении объектов, которые могут быть достаточно просто реализованы, т.е. имеют чётко определенную структуру и поведение.