## **1.1 Что такое инженерия и назначение инженера**

Производство любого изделия, строительство объекта любого назначения требует специалистов разного уровня: проектировщиков (или конструкторов), инженеров, мастеров, рабочих. И конечно же управленцев и снабженцев. Функции рабочих, как правило, связаны с повторением некоторых достаточно однотипных действий, которые несомненно, могут быть исключительно важными и сложными. Особенно с внедрением автоматизации и роботизацией производства.

**Инженер** – это специалист, осуществляющий инженерную деятельность. В свою очередь, **инженерная деятельность** – область технической деятельности, включающая в себя ряд специализированных областей и дисциплин. Эта деятельность направлена на практическое приложение и применение научных, экономических, социальных и практических знаний с целью обращения природных ресурсов на пользу человека. Конечной целью инженерной деятельности являются **изобретение, разработка, создание, внедрение, обслуживание и/или улучшение техники, материалов или процессов**.

Современный этап развития инженерной деятельности характеризуется **системным подходом** к решению сложных научно-технических задач, обращением ко всему комплексу социальных, гуманитарных, естественных и технических дисциплин.

**Проектирование** как особый вид инженерной деятельности формируется в начале 20-го столетия и связано с первоначальной деятельностью чертёжников, необходимостью особого (точного и понятного) графического изображения замысла инженера для передачи исполнителям на производстве.

Различают «внешнее» и «внутреннее» проектирование. **Первое** направлено на проработку общей идеи системы, её исследование с помощью теоретических средств, созданных и имеющихся в соответствующей технической науке. Второе связано с созданием рабочих чертежей (технических и рабочих проектов). Они служат основным документом для изготовления технической системы на производстве.

**Проектирование следует отличать от конструирования.** Продукт проектировочной деятельности выражается в особой знаковой форме – в виде текста, чертежей, графиков, расчётов, моделей в памяти ЭВМ и, как правило, технико–экономического обоснования. Результат конструкторской деятельности должен быть обязательно материализован в виде опытного образца. С его помощью уточняются расчёты, приводимые в проекте, и конструктивно-технические характеристики проектируемой технической системы.

Характерной чертой развития инженерной деятельности является специализация различных её видов. Сформировалась концепция CALS (Continuous Acquisition and Life cycle Support), суть которой состоит в применении принципов и технологий информационной поддержки на всех этапах (стадиях) производства (то есть жизненного цикла) изделия (ссылка 2).

Во второй половине 20-го века два фактора – возрастание сложности технических систем (вплоть до человеко-машинных и роботизированных), дифференциация инженерной деятельности привели к кризису традиционного инженерного мышления, которое было ориентировано на приложение знаний естественных и технических наук вкупе с созданием относительно простых технических систем. В результате сформировалась особая форма инженерной деятельности – схемотехническая, а также особой специальности: инженер-системотехник.

## **1.2 Особенности системотехнической деятельности**

Главной задачей инженера-системотехника является интеграция инженерной деятельности по отраслям и видам. Сама системотехническая деятельность является неоднородной и включает в себя различные виды инженерных разработок, реализуется различными группами специалистов, занимающихся разработкой отдельных подсистем. К ним относятся главный конструктор, руководитель темы, главный специалист проекта, руководитель научно-тематического отдела.

## **1.3 Социальная значимость результата инженерной деятельности**

В результате, отдельный инженер концентрирует свое внимание лишь на части сложной технической системы, а не на целом. При этом инженер (коллектив инженеров) все более и более удаляется от непосредственного потребителя его изделия, конструируя артефакт (техническую систему) отделенным от конкретного человека, служить которому прежде всего и призван инженер.

Это привело к возникновению **социотехнического проектирования**, необходимости подготовки специалистов соответствующего профиля.

В конечном счёте в инженерии сформировалась методология жизненного цикла изделия как совокупности всех существенных этапов «жизни» продукции рис.1.1.



Рис.1.1 Схематическое изображение полного жизненного цикла продукта изделия

## **1.4 Сущность и логика становления программной инженерии**

Программная инженерия связана с программированием.

На первых порах программисты в индивидуальном порядке решали небольшие научно-технические задачи на конкретной ЭВМ в числовом коде, а затем в автокоде или Ассемблере.

Понемногу задачи становились всё более сложными. Появились библиотеки алгоритмов, сложные языки программирования и методики тестирования. Эти задачи могли быть решены только коллективами программистов с чёткой организацией работы и разделением функций. Даже понимание заказчика и согласование с ним его требований требовало специальных знаний. То есть понадобились специальные системные подходы.

Производство программ стало бизнесом, который обязан быть прибыльным. Поэтому командам-производителям программ понадобился грамотный финансовый менеджмент и маркетинг. Создание программ стало требовать системного подхода, и как ответ на это требование сформировалась отрасль знаний под названием «программная инженерия». С самого начала у неё была одна особенность: международный характер информационных обменов обеспечил международный же характер разработок при создании стандартов, областей знаний и методик, составляющих ядро программной инженерии.

В заключение этого раздела приведём современные представления о масштабах и характеристиках программного обеспечения, а также классификациях программных проектов по его созданию (***Проект ПО* – совокупность спецификаций ПО (включающих модели и проектную документацию), обеспечивающих создание ПО в конкретной программно-технической среде**).

ПО можно разбить на два класса: «малое» и «большое».

*«****Малое» (простое****) программное обеспечение* имеет следующие характеристики:

● решает **одну несложную, четко поставленную задачу**;

**● размер исходного кода** не превышает **нескольких сотен строк**;

● **скорость** работы программного обеспечения и необходимые ему **ресурсы не играют большой роли**;

● **ущерб** от неправильной работы не имеет большого значения;

● **модернизация** программного обеспечения, дополнение его возможностей требуется редко;

**● как правило**, разрабатывается одним программистом или небольшой группой (**5 или менее человек**);

● **подробная документация** не требуется, ее может заменить исходный **код**, который **доступен**.

***«Большое» (сложное****) программное обеспечение* имеет **2-3 или более** характеристик из следующего перечня:

● решает **совокупность** взаимосвязанных задач;

● использование приносит **значимую выгоду**;

● **удобство** его использования играет **важную роль**;

● **обязательно** наличие **полной и понятной документации**;

● **низкая скорость** работы приводит к **потерям**;

● сбои, **неправильная работа**, наносит **ощутимый ущерб**;

● программы в составе ПО во время работы **взаимодействует** с другими **программами** и **программно-аппаратными комплексами**;

● работает на **разных** платформах;

● требуется **развитие**, **исправление ошибок**, добавление новых возможностей;

● группа разработчиков состоит из **более 5 человек**.

Классификация программных проектов по созданию сложного ПО может быть проведена по размеру группы разработчиков и длительности проекта:

● ***небольшие проекты*** – проектная команда менее 10 человек, срок от 3 до 6 месяцев;

● ***средние проекты*** – проектная команда от 20 до 30 человек, протяженность проекта 1-2 года;

● ***крупномасштабные проекты*** – проектная команда от 100 до 300 человек, протяженность проекта 3-5 лет;

● ***гигантские проекты***– **армия** разработчиков от 1000 до 2000 человек и более (включая консультантов и соисполнителей), протяженность проекта от 7 до 10 лет.

Эти характеристики проектов подтверждают приведённое выше заключение о социальной значимости соответствующих проектов и необходимости системного подходов в проектировании ПО, который должен быть основой проектирования программного обеспечения. Напомним, что ***системный подход*** – это методология исследования объекта любой природы как системы, а ***система***– это совокупность взаимосвязанных частей, работающих совместно для достижения некоторого результата. Определяющий признак системы – поведение системы в целом не сводимо к совокупности поведения частей системы, что является характерным свойством крупных проектов по разработке ПО.

## **1.5 Предпосылки возникновения программной инженерии**

В конце 60-х – начале 70-х годов прошлого века произошло событие, которое вошло в историю как **первый кризис программирования**. Событие состояло в том, что стоимость программного обеспечения стала приближаться к стоимости аппаратуры («железа»), а динамика роста этих стоимостей позволяла прогнозировать, что к середине 90-годов все человечество будет заниматься разработкой программ для компьютеров. Тогда и заговорили о программной инженерии (или технологии программирования, как это называлось в России) как о некоторой дисциплине, целью которой является сокращение стоимости программ. К настоящему времени программная инженерия накопила значительный багаж методологий, методик, стандартов и рекомендаций. Тем не менее, нельзя утверждать, что кризисные явления преодолены. Это выражается в том, что большие проекты выполняются с превышением сметы расходов и/или сроков, отведенных на разработку, а разработанное ПО не обладает требуемыми функциональными возможностями, имеет низкую производительность и качество. Анализом эффективности работы компаний по разработке ПО и эффективности их внедрения занимаются многие консалтинговые и аудиторские фирмы.

Одним из факторов неудачного создания ПО является плохое планирование, в результате устанавливаются невыполнимые сроки, закладываются недостаточные ресурсы. При этом основной причиной неверного планирования является заблуждение относительно производительности проектировщиков. В большом проекте общая производительность группы разработчиков не равна сумме производительностей отдельных членов группы (посчитанной, как если бы они работали в одиночку).

Отметим **особенности современных проектов ПО:**

● сложность – неотъемлемая характеристика создаваемого ПО;

● отсутствие полных аналогов и высокая доля вновь разрабатываемого ПО;

● наличие унаследованного ПО и необходимость его интеграции с разрабатываемым ПО;

● территориально распределенная и неоднородная среда функционирования;

● большое количество участников проектирования, разобщенность и разнородность отдельных групп разработчиков по уровню квалификации и опыту.

Разработка ПО имеет следующие **специфические** особенности:

● **неформальный** характер требований к ПО и формализованный основной объект разработки – программы;

● **творческий характер** разработки;

● **дуализм ПО**, которое, с одной стороны, является статическим объектом – совокупностью текстов, с другой стороны, – динамическим, поскольку при эксплуатации порождаются процессы обработки данных;

● при своем **использовании** (эксплуатации) ПО не **расходуется и не изнашивается, но морально устаревает**;

● «**неощутимость**», «воздушность», «квазинематериальность» ПО, что подталкивает к **безответственному переделыванию**, поскольку легко стереть и переписать, чего не сделаешь при проектировании зданий и аппаратуры.

Ответом на кризис в разработке ПО стало создание программной инженерии (software engineering) как специальной дисциплины или области знаний.

***Инженерия ПО*** (**software engineering**) – совокупность инженерных методов и средств создания ПО.

**Фундаментальная идея программной инженерии: проектирование ПО является формальным процессом, который можно изучать и совершенствовать.**

Основными целями программной инженерии являются:

● Системы должны создаваться в короткие сроки и соответствовать требованиям заказчика на момент внедрения.

● Качество ПО должно быть высоким.

● Разработка ПО должна быть осуществлена в рамках выделенного бюджета.

● Системы должны работать на оборудовании заказчика, а также взаимодействовать с имеющимся ПО.

● Системы должны быть легко сопровождаемыми и масштабируемыми.

Освоение и правильное применение методов и средств программной инженерии позволяет повысить качество, обеспечить управляемость процесса проектирования, что и является задачей программной инженерии как дисциплины, которую должны освоить проектировщики ПО.

С начала возникновения программная инженерия прошла достаточно длинный и интересный путь. Каждый этап связан с **появлением** (**или осознанием**) очередной проблемы и нахождением путей и способов решения этой проблемы.

## **1.6 Этапы развития программной инженерии**

### ***1.6.1 Эволюция технологий создания программного обеспечения***

Можно выделить три основных этапа становления технологии:

* 70-е и 80-е годы XX века– систематизация и стандартизация процессов создания ПО (структурный подход);
* 90-е годы- начало 21-го века – переход к сборочному, индустриальному способу создания ПО (объектно-ориентированный подход);
* с середины 90-х годов до настоящего времени – развитие компонентного подхода и сетевых технологий, создание CASE-технологий проектирования ПО

В рамках этих этапов развитие программной инженерии происходило и происходит по многим направлениям (нитям), решая проблемы, возникающие при разработке программного обеспечения:

1. [Императивное программирование](https://ru.wikipedia.org/wiki/Императивное_программирование) (императивная парадигма)
2. [Декларативное программирование](https://ru.wikipedia.org/wiki/Декларативное_программирование) (декларативная парадигма)
3. [Функциональное программирование](https://ru.wikipedia.org/wiki/Функциональное_программирование)
4. [Логическое программирование](https://ru.wikipedia.org/wiki/Логическое_программирование)
5. [Структурное программирование](https://ru.wikipedia.org/wiki/Структурное_программирование)
6. Модульное программирование
7. [Функциональное программирование](https://ru.wikipedia.org/wiki/Функциональное_программирование)
8. [Логическое программирование](https://ru.wikipedia.org/wiki/Логическое_программирование)
9. [Объектно-ориентированное программирование](https://ru.wikipedia.org/wiki/Объектно-ориентированное_программирование)
10. [Программирование, основанное на классах](https://ru.wikipedia.org/wiki/Объектно-ориентированное_программирование#_blank)
11. [Программирование, основанное на прототипах](https://ru.wikipedia.org/wiki/Прототипное_программирование)
12. [Субъектно-ориентированное программирование](https://ru.wikipedia.org/wiki/Субъектно-ориентированное_программирование)

1. Императивное программирование – это такой стиль написания исходного кода компьютерной программы, для которого характерно следующее:

* в исходном коде программы записываются инструкции (команды);
* инструкции выполняются последовательно;
* при выполнении инструкции данные, полученные при выполнении предыдущих инструкций, могут читаться из памяти;
* данные, полученные при выполнении инструкций, могут записываться в память

Императивная программа представляет собой последовательность команд, которые должен выполнить компьютер. Широко используется присваивание. Наличие операторов присваивания увеличивает сложность модели вычислений и делает императивные программы подверженными специфическим ошибкам, не встречающимся при функциональном подходе.

Рассмотрим развитие технологий программирования в рамках императивной парадигмы и ряд фундаментальных проблем разработки программ и найденных базовых методов их решения.

***Первый этап*** *– «стихийное» программирование*(от появления первых вычислительных машин до середины 60-х годов XX в). Первые программы имели простейшую структуру. Они состояли из собственно программы на машинном языке и обрабатываемых ею данных рис 1.2.



Рис 1.2. Структура первых программ

Появление *ассемблеров* позволило использовать символические имена данных и мнемоники кодов операций. В результате программы стали более «читаемыми».

Создание языков программирования высокого уровня, существенно упростило программирование вычислений, снизив уровень детализации операций, что позволило увеличить сложность программ.

Появление в языках средств, которые могли оперировать **подпрограммами** (отдельными блоками программного кода), позволило создать огромные библиотеки расчетных и служебных подпрограмм, которые можно было вызвать из разрабатываемой программы рис. 1.3.

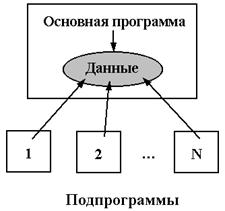


Рис.1.3. Архитектура программы с глобальной областью данных

Проблема была в том, что при увеличении количества подпрограмм возрастала вероятность искажения части глобальных данных какой-либо подпрограммой. Чтобы это исправить, было предложено в **подпрограммах размещать локальные данные** рис. 1.4.

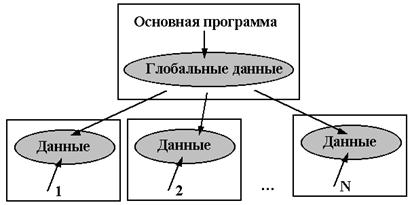


Рис 1.4. Архитектура программы использующей подпрограммы с локальными данными

Появление средств поддержки подпрограмм позволило осуществлять разработку программного обеспечения нескольким программистам параллельно.

Как отмечалось выше, в начале 60-х разразился «кризис программирования». Проект устаревал прежде, чем был готов к внедрению, увеличивалась его стоимость, и в результате многие проекты так и не были завершены.

Основная причина была в основном в том, что использовался подход «снизу-вверх» (***восходящее проектирование***). Интерфейсы подпрограмм получались сложными, и при сборке программного продукта выявлялось большое количество ошибок согласования. Процесс тестирования и отладки программ занимал более 80% времени разработки.

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.). В основе структурного подхода лежит *декомпозиция* (разбиение на части) сложных систем с целью последующей реализации в виде отдельных небольших подпрограмм. Проектирование осуществляется «сверху вниз» и подразумевает реализацию общей идеи, обеспечивая проработку интерфейсов подпрограмм (*нисходящее проектирование*).

Поддержка принципов структурного программирования была заложена в основу так называемых *процедурных языков программирования*. Среди наиболее известных языков этой группы стоит назвать PL/1, ALGOL-68, Pascal, С.

**Структурное программирование**— методология разработки программного обеспечения, в основе которой лежит представление программы в виде иерархической структуры блоков.

В соответствии с данной методологией любая программа строится без использования оператора goto из трёх базовых управляющих структур:

* **последовательность**,
* **ветвление**,
* **цикл**;
* кроме того, используются **подпрограммы**.

В 1970-е годы объёмы и сложность программ достигли такого уровня, что традиционная (неструктурированная) разработка программ перестала удовлетворять потребностям практики. Программы становились слишком сложными, чтобы их можно было нормально сопровождать. **Поэтому потребовалась систематизация процесса разработки и структуры программ**.

**Цель структурного программирования** — повысить производительность труда программистов, в том числе при разработке больших и сложных программных комплексов, сократить число ошибок, упростить отладку, модификацию и сопровождение программного обеспечения.

Таким образом была (до известной степени) решена проблема возрастания сложности программных комплексов.

К числу таких сложных программ относятся: системы управления космическими объектами, управления оборонным комплексом, автоматизации крупного финансового учреждения и т.д.

Для таких сложных программ оказалось, что **основная часть их стоимости приходится не на создание программ, а на их внедрение и эксплуатацию**. По аналогии с промышленной технологией стали говорить о **жизненном цикле** программного продукта, как о **последовательности определенных этапов: этапа проектирования, разработки, тестирования, внедрения и сопровождения.**

Основные принципы и идеи структурного программирования:

1. Алгоритм и программа должны разрабатываться поэтапно, по шагам. Этот принцип в литературе называется по-разному: *метод (принцип) последовательного построения (уточнения) алгоритма; принцип поэтапной детализации алгоритма; метод нисходящего (сверху вниз) проектирования.*
2. Логика любой программы должна опираться на минимальное количество следующих достаточно простых **базовых** (основных, типовых) управляющих (**алгоритмических**, логических) **структур** (конструкций) (БАС): **ветвление** (или развилка); **повторение** (или цикл); **следование**. Первая из структур программируется с помощью полной и сокращённой формы оператора *if* и вспомогательного оператора выбор (*switch* на языке *Си*, *case* в *Pascal*и др.). Основными операторами цикла являются оператор цикла с предусловием (*while*) и с постусловием (*do … while* на языке *Си*, *repeat … until* в *Pascal*и др.). Во всех языках программирования команды выполняются последовательно в том порядке, как они записаны, если не используются операторы, меняющие этот естественный порядок.
3. **Важным и самым сложным** является следующий принцип. **Каждая из БАС должна иметь один вход и один выход**. Поэтому структурное программирование в литературе часто называют программированием без оператора *goto,*который нарушает это требование*.*

И ещё одно правило, которое должно соблюдаться проектировщиками ПО. Должна соблюдаться строгая дисциплина планирования и документирования, поддержка соответствие кода проектной документации.

**Следующей проблемой**, тесно связанной с предыдущей, является **необходимость использования повторного использования кода**.

Дальнейший рост сложности и размеров разрабатываемого программного обеспечения потребовал развития *структурирования данных*, в языках появляется возможность определения пользовательских типов данных. Стремление разграничить доступ к глобальным данным программы, чтобы уменьшить количество ошибок, возникающих при работе с ними, привело к возникновению технологии модульного программирования.

Уже на первых этапах становления программной инженерии (даже когда она так еще не называлась) было отмечено, что высокая стоимость программ связана с разработкой одинаковых (или похожих) фрагментов кода в различных программах. **Использование при создании новых программ ранее написанных фрагментов сулило существенное снижение сроков и стоимости разработки**.

В связи с этим был выдвинут и реализован принцип модульного программирования.

**Модульное программирование**. Главный принцип модульного программирования состоял в выделении таких фрагментов и оформлении их в виде модулей. **Каждый модуль снабжался описанием, в котором устанавливались правила его использования** – **интерфейс модуля**.

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 и т.п. (рис. 1.5).

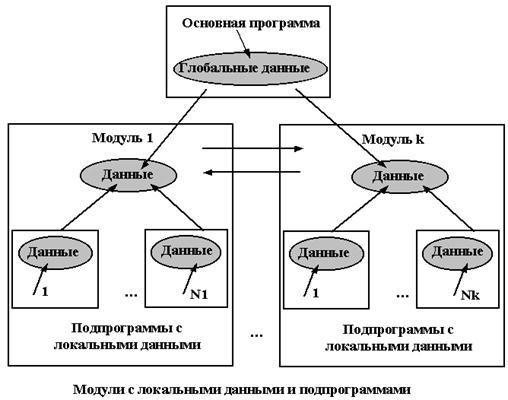


Рис.1.5. Архитектура программы, состоящей из модулей

Интерфейс задавал связи модуля с основной программой – **связи по данным** и **связи по управлению**. Связи между модулями при использовании данной технологии осуществляются через специальный интерфейс, в то время как доступ к реализации самого модуля (телам подпрограмм и некоторым «внутренним» переменным) запрещен. Эту технологию поддерживают современные версии языков Pascal и С (C++), языки Ада и Modula.

При таком подходе разработка программного обеспечения несколькими программистами значительно упрощается. Каждый разрабатывает свои модули независимо, обеспечивая взаимодействие через специально оговоренные межмодульные интерфейсы.

**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).

Наиболее простыми в этом отношении оказались модули решения математических задач: решения уравнений, систем уравнений, задач оптимизации. К настоящему времени накоплены и успешно используются большие библиотеки таких модулей.

Но для многих задач возможность их повторного использования оказалась проблематичной в виду сложности их связей с основной программой. Например, модуль расчета зарплаты, написанный для одной фирмы, может не подойти для другой, т.к. зарплата в этих фирмах рассчитывается не во всем одинаково.

Повторное использование модулей со сложными интерфейсами является достаточно актуальной и по сей день. Для ее решения разрабатываются специальные формы (структуры) представления модулей и организации их интерфейсов.

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 (объектно-ориентированный) подход**. Родоначальниками этого подхода были Буч(Booch)[[1]](#footnote-1) и Рамбо (Rumbaugh)[[2]](#footnote-2)

Теоретические разработки и внедрение этого подхода составляет сущность **третьего этапа** развития технологий проектирования, активное развитие которого приходится на период с середины 80-ых до конца 90-ых годов 20-го столетия.

Неизбежность появления этой технологии была обусловлена **продолжающимся ростом стоимости программного обеспечения**. Он был обусловлен тем, что что изменение требований к программе стали возникать не только на стадии сопровождения, но и на стадии проектирования – проблема заказчика, который не знает, что он хочет. В основном данная ситуация возникла, когда программное обеспечение стало эффективно использоваться в коммерческой и производственной деятельности предприятий.

Объектная структура программы впервые была использована в языке имитационного моделирования сложных систем Simula (60-е годы XX в.), в специализированном языке моделирования Smalltalk (70-е годы XX в.), а затем в новых версиях универсальных языков программирования, таких, как Pascal, C++, Modula, Java.

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* (рис. 1.6).

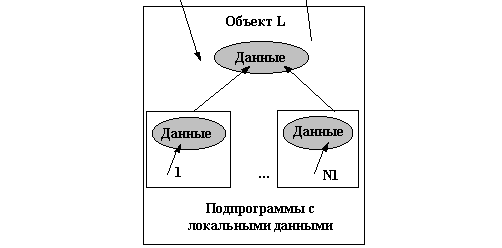
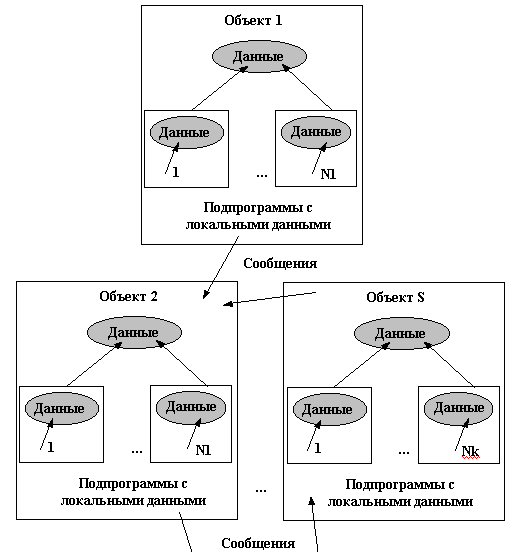


Рис.1.6. Архитектура программы при объектно-ориентированном программировании

Суть подхода состоит в том, что вводится **понятие класса как развитие понятия модуля** с определенными свойствами и поведением, характеризующими обязанностями класса. Каждый класс может порождать объекты – экземпляры данного класса. При этом работают основные принципы (парадигмы) ООП:

1. Инкапсуляция – объединение в классе данных (свойств) и методов (процедур обработки).
2. Наследование – возможность вывода нового класса из старого с частичным изменением свойств и методов
3. Полиморфизм – определение свойств и методов объекта по контексту

Основным достоинством объектно-ориентированного программирования по сравнению с модульным 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. Механизмы *наследования*, *полиморфизма*, *инкапсуляции* позволяют конструировать сложные объекты из сравнительно простых. В результате существенно увеличивается показатель повторного использования кодов и появляется возможность создания библиотек классов для различных применений.

Были созданы среды, поддерживающие визуальное программирование, например, Delphi, C++ Builder, 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, C++, Java, C# и других имеет существенные недостатки, к которым относятся:

* необходимость разработки программного обеспечения с использованием средств и возможностей ***одного*** языка программирования высокого уровня и ***одного*** компилятора;
* наличие исходных кодов используемых библиотек классов, так как 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.

**Современная технология программирования** — **компонентный подход,** который 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**. 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.

Технология СОМ фирмы Microsoft является развитием технологии *OLE (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. Модификация СОМ, обеспечивающая передачу вызовов между компьютерами, называется *DCOM (Distributed COM — распределенная (то есть сетевая) СОМ)*. Она располагает двоичным интерфейсом и поэтому каждый раз при поддержке нового языка программирования отображения описания интерфейса на языке IDL (Interface Definition Language).

Активное развитие компонентного подхода началось с середины 90-ых годов 20-го столетия и продолжается до нашего времени. Этот период считается **четвёртым** **этапом** развития технологии программирования.

**Компонент представляет собой готовый программный продукт**, который можно использовать как отдельно, так и совместно с подобными элементами в рамках решаемой задачи. В рамках этого подхода программное обеспечение строится из отдельных компонентов, физически отдельно существующих программных частей, которые распространяются в двоичном виде (в отличие от классов), взаимодействуют между собой посредством стандартизируемых интерфейсов и могут быть используемы в различных языках программирования (рис. 1.7 (1.6 из книги))

Таковы, вкратце, основные этапы изменения технологий программирования. Поскольку программное обеспечение на сегодняшний день используется в самых различных устройствах и сферах деятельности человека, то можно прогнозировать дальнейшее совершенствование технологий программирования. В частности, активно развивается технология облачных сервисов.

### ***1.6.2 Становление CASE-технологий***

Одновременно с развитием компонентного подхода развивались и внедрялись в практику так называемые CASE (Computer Aided Software Engineering) технологии проектирования программного обеспечения информационных систем, что оказалось новой ветвью в технологии промышленной разработки и реализации сложных и значительных по объему систем программного обеспечения.

Термин CASE 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 ИС в целом.**

Теперь под термином CASE-средства понимаются программные средства, поддерживающие процессы:

* + создания и сопровождения ИС, 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 полную среду разработки ИС.**

### ***1.6.3 Стандартизация и глобализация в сфере программной инженерии***

Развитие систем и средств вычислительной техники, телекоммуникационных систем и быстрое расширение сфер их применения привели к необходимости объединения конкретных вычислительных устройств и реализованных на их основе ИС в единые информационно-вычислительные системы и среды для формирования единого информационного пространства (Unified Information Area — UIA). Формирование такого пространства стало насущной необходимостью для решения многих важнейших экономических и социальных задач в ходе становления и развития информационного общества.

Такое пространство можно определить, как совокупность баз данных, хранилищ знаний, систем управления ими, информационно-коммуникационных систем и сетей**, методологий и технологий их разработки**, ведения и использования на основе единых принципов и общих правил, обеспечивающих информационное взаимодействие для удовлетворения потребностей пользователей. Основными составляющими единого информационного пространства являются:

При формировании единого информационного пространства менеджеры, архитекторы и разработчики программно-аппаратных средств столкнулись с рядом организационных, технических и технологических проблем. Например, разнородность технических средств вычислительной техники с точки зрения организации вычислительного процесса, архитектуры, систем команд, разрядности процессоров и шины данных потребовала создания стандартных физических интерфейсов, реализующих взаимную совместимость компьютерных устройств. Однако при дальнейшем увеличении числа типов интегрируемых устройств (число таких модулей в современных распределенных вычислительных и информационных системах исчисляется сотнями) сложность организации физического взаимодействия между ними существенно возрастала, что приводило к проблемам в управлении такими системами.

Поэтому достаточно быстро возникла необходимость стандартизации в сфере создания вычислительной техники по разным направлениям и создания программного обеспечения. Естественно, что появились стандарты в программной инженерии, работа над которыми с неизбежностью приняла международный характер. Ознакомление с такими стандартами является необходимым этапом в образовании проектировщиков ПО различных уровней и специализаций.

## **1.7 Заключение**

Программная инженерия как некоторое направление возникло и формировалось под давлением роста стоимости создаваемого программного обеспечения. Главная цель этой области знаний - сокращение стоимости и сроков разработки программ.

Программная инженерия прошла несколько этапов развития, в процессе которых были сформулированы фундаментальные принципы и методы разработки программных продуктов.

Основной принцип программной инженерии состоит в том, что программы создаются в результате выполнения нескольких взаимосвязанных этапов (анализ требований, проектирование, разработка, внедрение, сопровождение), составляющих жизненный цикл программного продукта.

Программная инженерия, как и любая другая, занимается не только техническими вопросами производства ПО (специфицирование требований, проектирование, кодирование, …), но и управлением программными проектами, включая **вопросы планирования, финансирования, управления коллективом и т.д**. Кроме того, задачей программной инженерии является разработка средств, методов и теорий для поддержки процесса производства ПО.

Фундаментальными методами проектирования и разработки являются модульное, структурное и объектно-ориентированное проектирование и программирование.

Несмотря на то, что программная инженерия достигла определенных успехов, перманентный кризис программирования продолжается. Связано это с взрывным ростом использования информационных средств: персональный компьютер, локальные и глобальные вычислительные сети, мобильная связь, электронная почта, семантический Интернет, интеллектуальные технологии, роботизация, Интернет вещей, обеспечение компьютерной безопасности и т.д.

1. [https://ru.wikipedia.org/wiki/%D0%91%D1%83%D1%87,\_%D0%93%D1%80%D0%B0%D0%B4%D0%B8](https://ru.wikipedia.org/wiki/Буч,_Гради) [↑](#footnote-ref-1)
2. [https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%BC%D0%B1%D0%BE,\_%D0%94%D0%B6%D0%B5%D0%B9%D0%BC%D1%81](https://ru.wikipedia.org/wiki/Рамбо,_Джеймс) [↑](#footnote-ref-2)