**Технологии разработки программ**

1. **Структурно-модульное программирование**

**Структу́рное программи́рование** — методология разработки [программного обеспечения](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), в основе которой лежит представление программы в виде иерархической структуры [блоков](https://ru.wikipedia.org/wiki/%D0%91%D0%BB%D0%BE%D0%BA_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29). Предложена в 1970-х годах [Э. Дейкстрой](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%B9%D0%BA%D1%81%D1%82%D1%80%D0%B0,_%D0%AD%D0%B4%D1%81%D0%B3%D0%B5%D1%80_%D0%92%D0%B0%D0%B9%D0%B1) и др.

В соответствии с данной методологией любая программа строится без использования оператора goto из трёх базовых управляющих структур: последовательность, ветвление, цикл; кроме того, используются [подпрограммы](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B4%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0). При этом разработка программы ведётся пошагово, методом «сверху вниз».

**Принципы структурного программирования**

*1.* **Следует отказаться от использования оператора безусловного перехода goto.**

*2.* **Любая программа строится из трёх базовых управляющих конструкций: последовательность, ветвление, цикл.**

* Последовательность — однократное выполнение операций в том порядке, в котором они записаны в тексте программы.
* Ветвление — однократное выполнение одной из двух или более операций, в зависимости от выполнения заданного условия.
* Цикл — многократное исполнение одной и той же операции до тех пор, пока выполняется заданное условие (условие продолжения цикла).

*3.* **В программе базовые управляющие конструкции могут быть вложены друг в друга произвольным образом.** Никаких других средств управления последовательностью выполнения операций не предусматривается.

*4.* **Повторяющиеся фрагменты программы можно оформить в виде** [***подпрограмм***](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B4%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0) **(процедур и** [**функций**](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29)**).** Таким же образом (в виде подпрограмм) можно оформить логически целостные фрагменты программы, даже если они не повторяются.

* В этом случае в тексте основной программы, вместо помещённого в подпрограмму фрагмента, вставляется инструкция «Вызов подпрограммы». При выполнении такой инструкции работает вызванная подпрограмма. После этого продолжается исполнение основной программы, начиная с инструкции, следующей за командой «Вызов подпрограммы».
* Бертран Мейер поясняет: «Преобразуйте элемент, возможно, с внутренними элементами, в подпрограмму, характеризуемую одним входом и одним выходом в [потоке управления](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%80%D1%8F%D0%B4%D0%BE%D0%BA_%D0%B2%D1%8B%D0%BF%D0%BE%D0%BB%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F#.D0.BF.D0.BE.D1.82.D0.BE.D0.BA_.D0.B2.D1.8B.D0.BF.D0.BE.D0.BB.D0.BD.D0.B5.D0.BD.D0.B8.D1.8F)»[[21]](https://ru.wikipedia.org/wiki/%D1%F2%F0%F3%EA%F2%F3%F0%ED%EE%E5_%EF%F0%EE%E3%F0%E0%EC%EC%E8%F0%EE%E2%E0%ED%E8%E5#cite_note-21).

*5.* **Каждую логически законченную группу инструкций следует оформить как** [**блок**](https://ru.wikipedia.org/wiki/%D0%91%D0%BB%D0%BE%D0%BA_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29) **([block](https://en.wikipedia.org/wiki/Block_%28programming%29" \o "en:Block (programming))).** Блоки являются основой структурного программирования.

Блок — это логически сгруппированная часть исходного кода, например, набор инструкций, записанных подряд в исходном коде программы. Понятие *блок* означает, что к блоку инструкций следует обращаться как к единой инструкции. Блоки служат для ограничения области видимости переменных и функций. Блоки могут быть пустыми или вложенными один в другой. Границы блока строго определены. Например, в if-инструкции блок ограничен кодом BEGIN..END (в языке Паскаль) или фигурными скобками {...} (в языке C) или отступами (в языке Питон).

*6.* **Все перечисленные конструкции должны иметь один вход и один выход.**.

*7.* **Разработка программы ведётся пошагово, методом «сверху вниз»** *(top–down method)*.

**Метод проектирования «сверху вниз»**

Сначала пишется текст основной программы, в котором, вместо каждого связного логического фрагмента текста, вставляется вызов подпрограммы, которая будет выполнять этот фрагмент. Вместо настоящих, работающих подпрограмм, в программу вставляются фиктивные части — *заглушки*, которые, говоря упрощенно, ничего не делают.

Если говорить точнее, заглушка удовлетворяет требованиям интерфейса заменяемого фрагмента (модуля), но не выполняет его функций или выполняет их частично. Затем заглушки заменяются или дорабатываются до настоящих полнофункциональных фрагментов (модулей) в соответствии с планом программирования. На каждой стадии процесса реализации уже созданная программа должна правильно работать по отношению к более низкому уровню. Полученная программа проверяется и отлаживается.

После того, как [программист](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%81%D1%82) убедится, что подпрограммы вызываются в правильной последовательности (то есть общая структура программы верна), подпрограммы-заглушки последовательно заменяются на реально работающие, причём разработка каждой подпрограммы ведётся тем же методом, что и основной программы. Разработка заканчивается тогда, когда не останется ни одной заглушки.

Такая последовательность гарантирует, что на каждом этапе разработки программист одновременно имеет дело с обозримым и понятным ему множеством фрагментов, и может быть уверен, что общая структура всех более высоких уровней программы верна.

При сопровождении и внесении изменений в программу выясняется, в какие именно процедуры нужно внести изменения. Они вносятся, не затрагивая части программы, непосредственно не связанные с ними. Это позволяет гарантировать, что при внесении изменений и исправлении ошибок не выйдет из строя какая-то часть программы, находящаяся в данный момент вне зоны внимания программиста.

**Достоинства структурного программирования**

Следование принципам структурного программирования обеспечило ясность и читаемость программ.

1. Структурное программирование позволяет значительно сократить число вариантов построения программы по одной и той же спецификации, что значительно снижает сложность программы и, что ещё важнее, облегчает понимание её другими разработчиками.
2. В структурированных программах логически связанные операторы находятся визуально ближе, а слабо связанные — дальше, что позволяет обходиться без [блок-схем](https://ru.wikipedia.org/wiki/%D0%91%D0%BB%D0%BE%D0%BA-%D1%81%D1%85%D0%B5%D0%BC%D0%B0) и других графических форм изображения алгоритмов (по сути, сама программа является собственной блок-схемой).
3. Сильно упрощается процесс [тестирования](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F) и [отладки](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BB%D0%B0%D0%B4%D0%BA%D0%B0) структурированных программ.

## Основные принципы структурной методологии.

### Принцип абстракции.

Этот принцип позволяет разработчику рассматривать программу в нужный момент без лишней детализации. Детализация увеличивается при переходе от верхнего уровня абстракции к нижнему.

### Принцип формальности.

Он предполагает строгий методический подход к программированию, придает творческому процессу определенную строгость и дисциплину

### Принцип модульности.

В соответствии с этим принципом программа разделяется на отдельные законченные фрагменты, модули, которые просты по управлению и допускают независимую отладку и тестирование. В результате отдельные ветви программы могут создаваться разными группами программистов.

### Принцип иерархического упорядочения.

Взаимосвязь между частями программы должна носить иерархический, подчиненный характер. Это, кстати, следует и из принципа нисходящего проектирования.

## Нисходящее проектирование.

Нисходящее проектирование строится на вышеперечисленных принципах. При нисходящем проектировании происходит анализ задачи с целью определения возможности разбиения ее на ряд подзадач. Затем каждая из полученных подзадач также анализируется для возможного разбиения на подзадачи. Процесс для очередной подзадачи заканчивается, когда подзадачу невозможно или нецелесообразно разбивать на подзадачи далее. Результат этот процесса, зафиксированный в графической форме, является основой для построения структурной схемы программы, которая показывает, во-первых, что делает вся программа в целом и ее отдельные части, а, во-вторых, отображает взаимосвязь подзадач друг с другом.

На основе структурной схемы программы выполняется реализация подзадач в виде отдельных модулей

После разбиения программного комплекса на программные модули и подготовки спецификаций на каждый программный модуль начинается работа по проектированию алгоритмов, реализующих спецификацию каждого модуля

## Структурное кодирование.

Структурноe кодирование - это метод кодирования (программирования), предусматривающий создание понятных, простых и удобочитаемых программных модулей и программных комплексов на требуемых языках программирования.

Для кодирования программных модулей используются унифицированные (базовые) структуры. Доказано, что любая программа может быть составлена с применением только трёх канонических структур. Программные комплексы и программные модули, закодированные в соответствии с правилами структурного программирования, называются структурированными.

## Модульное программирование.

Модульное программирование - это организация программы как совокупности небольших независимых блоков, модулей, структура и поведение которых подчиняется определенным правилам. Следует заметить, что понятие «модуль» не совпадает в данном случае с понятием «модуль» ( в смысле «библиотека») языка Паскаль. Это должна быть простая, замкнутая (независимая) программная единица (процедура или функция), обозримая, реализующая только одну функцию. Для написания одного модуля должно быть достаточно минимальных знаний о тексте других, как вызывающих, так и вызываемых.

Программа, разработанная в соответствии с принципами структурного программирования, должна удовлетворять следующим требованиям:

* программа должна разделяться на независимые части, называемые модулями
* модуль - это независимый блок, код (текст) которого физически и логически отделен от кода других модулей;
* модуль выполняет только одну логическую функцию, иначе говоря, должен решать самостоятельную задачу своего уровня по принципу: один программный модуль - одна функция;
* работа программного модуля не должна зависеть:

от входных данных;

от того, какому программному модулю предназначены его выходные данные;

от предыстории вызовов программного модуля;

* размер программного модуля желательно ограничивать одной-двумя страницами исходного листинга (50-100 строк исходного кода);
* модуль должен иметь только одну входную и одну выходную точку;
* взаимосвязи между модулями устанавливаются по иерархической структуре;
* каждый модуль должен начинаться с комментария, объясняющего его назначение, назначение переменных, передаваемых в модуль и из него, модулей, которые его вызывают, и модулей, которые вызываются из него;
* при создании модуля можно использовать только стандартные управляющие конструкции: выбор, цикл, блок (последовательность операторов);
* оператор безусловного перехода или вообще не используется в модуле, или применяется в исключительных случаях только для перехода на выходную точку модуля;
* в тексте модуля необходимо использовать комментарии, в особенности в сложных местах алгоритма;
* идентификаторы переменных и модулей должны быть смысловыми, «говорящими»;
* в одной строке стоит записывать не более одного оператора. Если для записи оператора требуется больше, чем одна строка, то все последующие операторы записываются с отступами;
* желательно не допускать вложенности более, чем трех уровней;
* следует избегать использования языковых конструкций с неочевидной семантикой и программистских «трюков».

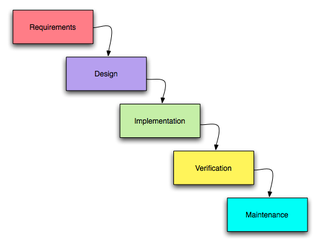
В заключение следует напомнить, что все эти вместе взятые меры направлены на повышение качества разрабатываемого программного обеспечения.

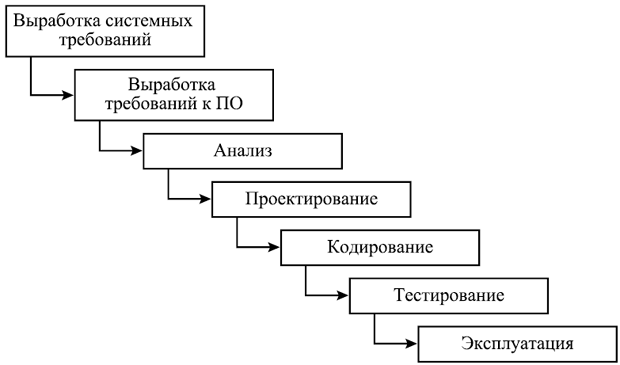
1. **Модель водопада разработки ПО**

**Каскадная модель** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *waterfall model*, иногда переводят, как *модель "Водопад"*) — [модель](https://ru.wikipedia.org/w/index.php?title=%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%28%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0%29&action=edit&redlink=1) процесса [разработки программного обеспечения](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F), в которой процесс разработки выглядит как поток, последовательно проходящий фазы анализа требований, проектирования, реализации, тестирования, интеграции и поддержки

В оригинальной каскадной модели Ройса, следующие фазы шли в таком порядке:

1. Определение требований
2. Проектирование
3. Конструирование (также «реализация» либо «кодирование»)
4. Воплощение
5. Тестирование и отладка (также «верификация»)
6. Инсталляция
7. Поддержка

[](https://commons.wikimedia.org/wiki/File:Waterfall_model.png?uselang=ru)

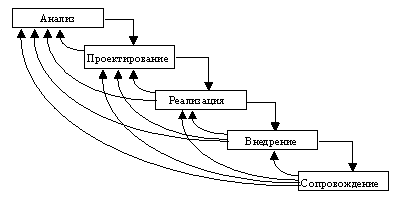


Переход от одной фазы к другой происходит только после полного и успешного завершения предыдущей

Следуя каскадной модели, разработчик переходит от одной стадии к другой строго последовательно. Сначала полностью завершается этап «определение требований», в результате чего получается список требований к ПО. После того как требования полностью определены, происходит переход к проектированию, в ходе которого создаются документы, подробно описывающие для программистов способ и план реализации указанных требований. После того как проектирование полностью выполнено, программистами выполняется реализация полученного проекта. На следующей стадии процесса происходит интеграция отдельных компонентов, разрабатываемых различными командами программистов. После того как реализация и интеграция завершены, производится тестирование и отладка продукта; на этой стадии устраняются все недочёты, появившиеся на предыдущих стадиях разработки. После этого программный продукт внедряется и обеспечивается его поддержка — внесение новой функциональности и устранение ошибок.

Тем самым, каскадная модель подразумевает, что переход от одной фазы разработки к другой происходит только после полного и успешного завершения предыдущей фазы, и что переходов назад либо вперёд или перекрытия фаз — не происходит.

Методику «Каскадная модель» довольно часто критикуют за недостаточную гибкость и объявление самоцелью формальное управление проектом в ущерб срокам, стоимости и качеству. Тем не менее, при управлении большими проектами формализация часто являлась очень большой ценностью, так как могла кардинально снизить многие [риски](https://ru.wikipedia.org/wiki/%D0%A0%D0%B8%D1%81%D0%BA) проекта и сделать его более [прозрачным](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B7%D1%80%D0%B0%D1%87%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%28%D0%BF%D0%BE%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5%29). Начиная с 2009 года, формально [Институтом управления проектами (PMI)](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%81%D1%82%D0%B8%D1%82%D1%83%D1%82_%D0%BF%D0%BE_%D0%A3%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8E_%D0%9F%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B0%D0%BC%D0%B8) предлагается как стандарт гибридный вариант методологии управления проектами, сочетающий в себе как плюсы от методики «Водопада», так и достижения итеративных методологов.



1. **Спиральная модель**

**Спира́льная модель**, предложенная Барри Боэмом в [1986 году](https://ru.wikipedia.org/wiki/1986_%D0%B3%D0%BE%D0%B4), стала существенным прорывом в понимании природы разработки ПО. Она представляет собой [процесс разработки программного обеспечения](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F), сочетающий в себе как проектирование, так и постадийное [прототипирование](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D1%82%D0%B8%D0%BF%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) с целью сочетания преимуществ восходящей и нисходящей концепции, делающая упор на начальные этапы жизненного цикла: анализ и проектирование. Отличительной особенностью этой модели является специальное внимание рискам, влияющим на организацию жизненного цикла. Боэм формулирует десять наиболее распространённых (по приоритетам) рисков:

1. Дефицит специалистов.
2. Нереалистичные сроки и бюджет.
3. Реализация несоответствующей функциональности.
4. Разработка неправильного пользовательского интерфейса.
5. «Золотая сервировка», перфекционизм, ненужная оптимизация и оттачивание деталей.
6. Непрекращающийся поток изменений.
7. Нехватка информации о внешних компонентах, определяющих окружение системы или вовлечённых в интеграцию.
8. Недостатки в работах, выполняемых внешними (по отношению к проекту) ресурсами.
9. Недостаточная производительность получаемой системы.
10. «Разрыв» в квалификации специалистов разных областей знаний.



Большая часть этих рисков связана с организационными и процессными аспектами взаимодействия специалистов в проектной команде.

Каждый виток спирали соответствует созданию фрагмента или версии программного обеспечения, на нем уточняются цели и характеристики проекта, определяется его качество и планируются работы следующего витка спирали. Таким образом углубляются и последовательно конкретизируются детали проекта и в результате выбирается обоснованный вариант, который доводится до реализации. Каждый виток разбит на 4 сектора:

* оценка и разрешение рисков,
* определение целей,
* разработка и тестирование,
* планирование.

На каждом витке спирали могут применяться разные модели процесса разработки ПО. В конечном итоге на выходе получается готовый продукт. Модель сочетает в себе возможности модели прототипирования и [водопадной модели](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%B2%D0%BE%D0%B4%D0%BE%D0%BF%D0%B0%D0%B4%D0%B0). Разработка итерациями отражает объективно существующий спиральный цикл создания системы. Неполное завершение работ на каждом этапе позволяет переходить на следующий этап, не дожидаясь полного завершения работы на текущем. При итеративном способе разработки недостающую работу можно будет выполнить на следующей итерации. Главная задача — как можно быстрее показать пользователям системы работоспособный продукт, тем самым активизируя процесс уточнения и дополнения требований. Основная проблема спирального цикла — определение момента перехода на следующий этап. Для ее решения необходимо ввести временные ограничения на каждый из этапов жизненного цикла. Переход осуществляется в соответствии с планом, даже если не вся запланированная работа закончена. План составляется на основе статистических данных, полученных в предыдущих проектах, и личного опыта разработчиков. Одним из возможных подходов к разработке программного обеспечения в рамках спиральной модели жизненного цикла является получившая в последнее время широкое распространение методология быстрой разработки приложений [RAD](https://ru.wikipedia.org/wiki/RAD_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29) (Rapid Application Development). Под этим термином обычно понимается процесс разработки программного обеспечения, содержащий 3 элемента:

1. небольшую команду программистов (от 2 до 10 человек);
2. короткий, но тщательно проработанный производственный график (от 2 до 6 месяцев);
3. повторяющийся цикл, при котором разработчики, по мере того, как приложение начинает обретать форму, запрашивают и реализуют в продукте требования, полученные через взаимодействие с заказчиком.

Жизненный цикл программного обеспечения по методологии RAD состоит из четырёх фаз:

1. фаза определения требований и анализа;
2. фаза проектирования;
3. фаза реализации;
4. фаза внедрения.

Спиральная модель ориентирована на большие, дорогостоящие и сложные проекты. В условиях, когда бизнес цели таких проектов могут измениться, но требуется разработка стабильной архитектуры, удовлетворяющей высоким требованиям по нагрузке и устойчивости, имеет смысл применение [Spiral Architecture Driven Development](http://sadd4ru.codeplex.com). Данная методология, включающая в себя лучшие идеи спиральной модели и некоторых других, позволяет существенно снизить архитектурные риски, что является немаловажным фактором успеха при разработке крупных систем.

1. Бизнес-моделирование

### Моделирование бизнес-процессов

 Что такое бизнес-процесс (БП)? Достаточно общее его представление: это преобразование входа в выход. Известны и другие подходы к заданию БП. Например, определение «Начинайте копать и копайте от забора до обеда». Это тоже задание некоторого процесса, только задание в своеобразной форме, когда последовательность, состав, время, логика работ не очень ясны. Если к этому описанию добавить, кто будет исполнять работу, как будет исполнять эту работу, в какой последовательности, какими документами воспользуется при исполнении работ, какую отчетность следует составлять в ходе исполнения работ, т. е. если процесс исполнения работ будет описан детально, подробно, документированно, то такое описание работ и называется БП. Это понимание дополняют приводимые далее ключевые понятия.  
    **Модель** – это отображение процесса, создаваемое для решения прикладных задач. Модель создается с помощью специализированного языка. Это могут быть язык графики, язык схем, таблицы или текстовые описания. Договоренность о том, как отображается процесс, с помощью какого языка, называется **нотацией описания** БП.  
   Выделяют **модели верхнего уровня** и **детальные модели процессов**. Речь идет о том, что описание работ можно сформировать агрегированно, и такое агрегированное описание называется моделью процессов верхнего уровня. Описание работ можно детализировать. Способ и уровень детализации зависят от предназначения конкретной детальной модели, поскольку модель бизнес-процесса – это прежде всего описание работ для решения прикладных задач. Какова прикладная задача, такова и детализация описания работ, соответственно такова и детализация описания БП.  
   Специалисты выделяют две типовые, распространенные нотации детализации, или два способа описаний БП. Первый способ связан с представлением процессов как **алгоритмов** исполнения работ, например, их **блок-схем** (состояние входа – преобразование – состояние выхода – логические условия). Такую нотацию удобно использовать, когда речь идет о необходимости регламентации логики исполнения деятельности, когда нужно объяснить, как реализуется процесс, кто это делает, в какой последовательности, какие документы создаются в ходе исполнения этой последовательности   
   Другая модель представления БП связана с описанием процесса как **потока объектов** (поток на входе – преобразование – поток на выходе). В качестве потоков могут выступать информация, документы, материальные поставки, другие ресурсы. Такие потоковые модели применяются в рамках рассмотрения отдельных задач и рассмотрения деятельности компании и ее подразделений в форме «вход-выход». На вход поступают ресурсы, на выходе получаем продукты (услуги). Именно для отслеживания того, что происходит с поставками от входа к выходу и служат процессные, потоковые модели процессов.

### Современные подходы к моделированию бизнес-процессов

Начало и окончание БП могут быть по-разному определены для одной и той же ситуации.

**Пример 1.** Бизнес-процесс «Разработка продукта». Начало и окончание – от требований клиента к продукту до создания конструкторской документации.  
    **Пример 2.** Бизнес-процесс «Закупка материальной ценности». Начало и окончание – от выбора поставщиков к отпуску товарно-материальных ценностей к производству.

   Представление процесса – это действие, это начало и окончание действия, это границы этого действия. Действие может рассматриваться как функция. При такой интерпретации БП включает функцию, начало и окончание действия функции.  
    **Назначение модели процесса.** Зачем нужны модели БП? Модель БП помогает понять, как устроена работа, позволяет регламентировать эту работу, т. е. зафиксировать порядок ее исполнения. Модель БП помогает управлять: если известен порядок исполнения работы, можем задавать ее параметры, планы, ресурсы, сроки исполнения работ, планировать эти параметры, обеспечивать организацию их исполнения, контролировать исполнение, регулировать ход исполнения (рис. 2.2.1).  
   Еще одно очень важное применение, или сфера, в которой модели БП очень помогают, – это сфера автоматизации. Описанный в виде моделей БП можно автоматизировать. Это касается потоков информации, которые возникают и используются в ходе исполнения БП, можно обеспечить исполнение отдельных работ или комплексов работ по ходу БП.  
   Поэтому модель – это описание БП на специализированном языке или его представление в виде таблиц, схем, текстов. Оно помогает понять процесс, управлять им, регламентировать и автоматизировать его.  
    **Вопросы, которые интересуют пользователей при моделировании процессов.** Специалисты, которые используют модели БП, – это прежде всего менеджеры, руководители и исполнители, включенные в организацию и исполнение БП. Что их интересует, какие главные вопросы, ответы на которые они хотят получить при использовании моделей БП? Модели БП помогают отвечать на такие вопросы (рис. 2.2.2):

##### Рис. 2.2.2. Вопросы, которые интересуют пользователей при моделировании бизнес-процессов

   • Какие работы необходимо выполнять для исполнения процесса в целом и достижения необходимого результата?  
   • Кто и какие работы выполняет? Каково распределение работ между исполнителями?  
   • Каков порядок, какова последовательность выполнения работ? Что выполняется сначала, что – потом, что – на финише.  
   • Что является результатом каждой из работ? Если выполняется несколько работ, которые связаны или интегрированы в цепочку работ, то что является выходом первой работы и входом второй?  
   • Какие ресурсы необходимы для выполнения работ?  
   Модели бизнес-процессов помогают ответить и на кардинальные вопросы повышения эффективности бизнеса. Как улучшить деятельность? Как повысить производительность? Как избежать проблем? Как повысить экономическую эффективность деятельности?  
   В определенном смысле можно сказать, что модель БП – это описание работы, выполняемой качественно и обеспечивающей достижение необходимой эффективности.  
    **Основная и обеспечивающая деятельность компании, управление и развитие.** Известны несколько базовых типологий, классификаций описания бизнес-процессов. Одна из наиболее распространенных в практике бизнеса типологий строится в зависимости от классификации областей деятельности, представленных на рис. 2.2.3. Деятельность компании подразделяется на текущую деятельность и деятельность по развитию. Текущая деятельность направлена на разработку, производство и предоставление продуктов (услуг) потребителю. А деятельность по развитию нацелена на создание будущих продуктов (услуг) и на улучшение деятельности организации в перспективе.  
   Текущая деятельность подразделяется на основную, в рамках которой осуществляется производство и предоставление продуктов (услуг) потребителям, и обеспечивающую, или поддерживающую.  
   В рамках поддерживающей деятельности обеспечивается необходимый внутренний сервис, производство продуктов (услуг), необходимых для основной деятельности.  
   Еще одна важная область деятельности компании – это управление основной деятельностью и управление обеспечивающей, поддерживающей деятельностью.

##### Рис. 2.2.3. Классификация областей деятельности компании (пример)

**Классификация бизнес-процессов: основные, поддерживающие, управление и развитие.** Исходя из представленной типологии описания БП можно более подробно охарактеризовать различные группы процессов (рис. 2.2.4).  
   Итак, **основные бизнес-процессы**:  
   • образуют добавленную стоимость продукта (услуги);  
   • создают продукт (услуги), представляющий ценность для клиента;  
   • формируют такой результат, такие потребительские качества, за которые клиент готов платить деньги.  
   • сфокусированы на получении прибыли.  
   Если потребителями результатов основных БП являются клиенты, внешние пользователи, то результаты поддерживающих БП потребляют бизнесы или основные БП.  
   Поэтому **поддерживающие бизнес-процессы**:  
   • направлены на предоставление необходимых внутренних продуктов, внутренних услуг для направлений бизнеса;  
   • обеспечивают функционирование инфраструктуры компании.  
    **Бизнес-процессы развития**:  
   • нацелены на получение прибыли в долгосрочной перспективе (не создают «прибыль сегодня»);  
   • обеспечивают не организацию текущей деятельности, а развитие или совершенствование деятельности компании в перспективе.  
    **Процессы управления:**   
   • нацелены на управление всеми тремя группами БП, т. е. управление основными БП, управление поддерживающими БП и управления БП развития.  
   Управление БП развития называют еще стратегическим управлением, т. е. стратегическое управление – это управление БП развития.

##### Рис. 2.2.4. Классификация бизнес-процессов

**Варианты развития бизнес-процессов.** Что происходит с БП? Какова их эволюция? Каков их жизненный цикл? В зависимости от обстоятельств, от стратегий, от условий внешней среды все может происходить по-разному (рис. 2.2.5).  
   Основные процессы являются источниками доходов, они определяют профиль бизнеса и имеют стратегическое значение для компании. Поэтому когда компания формирует свою стратегию, одним из главных блоков, образующих стратегию, является блок деловой стратегии, т. е. стратегии развития основных процессов.  
   Что может происходить с основными БП? Они могут поддерживаться в сформировавшемся состоянии, они могут развиваться или, наоборот, отмирать, в зависимости от требований рынка и стратегии компании. Поэтому основные процессы либо остаются как есть, либо развиваются, либо отмирают.  
   Поддерживающие процессы создают продукты и услуги для внутреннего потребления компании. При определенных обстоятельствах поддерживающие процессы и результаты деятельности от обеспечивающих, поддерживающих процессов могут направляться и вовне.

##### Рис. 2.2.5. Варианты развития бизнес-процессов

**Пример.** Многие внутренние инжиниринговые услуги, например, разработка технико-экономических обоснований, проектирование, организация поставок, контроль качества, выполнение различного рода высокотехнологичных услуг, оценка, структурирование проектов, – это те услуги, которые компания использует внутри для организации основной деятельности. При выводе их на достаточно высокий уровень качества и эффективности исполнения они могут быть востребованы вне компании и представляться на рынок.

   У каждого поддерживающего процесса есть еще одна альтернатива, кроме классической судьбы (оставаться и совершенствоваться как внутренний процесс либо развиваться в качестве внутреннего процесса, либо отмирать). Процесс может превратиться из поддерживающего в процесс двойного назначения – поддерживать внутренние услуги и предоставлять продукцию (услуги) вовне. Поэтому в зависимости от стратегии компании определенная часть поддерживающих процессов может превращаться в основные. Таким образом, не только БП, не только потребители влияют на судьбу поддерживающих процессов, но и рынок. Так, поддерживающие процессы могут отмирать в случае конкурентоспособных альтернатив на рынке, т. е. компания может поддерживать получение необходимых услуг не от внутренних служб, а от внешних компаний на условиях аутсорсинга.  
   
   Разные группы бизнес-процессов исполняются подразделениями с необходимой профильной специализацией. В практике бизнеса могут применяться нотации описания бизнес-процессов, позиционирующие на организационной схеме подразделения в зависимости от типологии исполняемых бизнес-процессов (рис. 2.2.6).

##### Рис. 2.2.6. Пример представления организационной схемы компании с привязкой к типологии бизнес-процессов

### 2.3. Корневая модель бизнес-процессов и ее использование [3]

##### Рис. 2.3.1. Направления использования корневой модели бизнес-процессов

   С чего надо начинать описание бизнес-процессов? Практика сформировала три подхода, или два варианта ответа на этот вопрос (рис. 2.3.2).  
    **Вариант 1.** Начинать с пилотной зоны, выбрать некоторый ограниченный приоритетный процесс, задать границы описания и моделирования, детально описать, попробовать, научиться описывать бизнес-процессы, оценивать полученный результат и на его основе получать решение о порядке проведения работ. Это подход к моделированию снизу. От начального объекта проводится последовательное расширение зоны описания и моделирования.  
    **Вариант 2.** Предполагает проведение описания сверху: сначала составляют модель процессов верхнего уровня, определяют ее компоненты, а потом отдельные компоненты детализируют в зависимости от направления их применения.  
    **Вариант 3.** Описывать итерационно снизу (детально) – вверх (агрегированно), а потом наоборот.  
   Реализация подхода «описываем сверху – от корневой модели БП» позволяет попутно решить следующие задачи и удовлетворить связанные с ними требования:  
   • системно, агрегированно представить организацию деятельности всей компании – корневая модель БП дает описание основных работ и представление о том, как эти работы увязаны между собой (см. рис. 2.3.1);  
   • наглядно показать распределение зон ответственности между подразделениями компании за исполнение основных работ (модель распределения основных зон ответственности);  
   • увязать стратегию и процессы между собой (установить соответствие между стратегией компании и процессами, которые ее поддерживают и в ходе реализации которых реализуется стратегия);  
   • «оцифровать» стратегию, для этого необходимо представить не только направления деятельности, но и с помощью количественных целевых показателей проследить проекцию стратегии на процессы, можно построить и проекцию количественных показателей на процессы (каскадировать показатели результативности). В результате модель верхнего уровня позволяет определить ключевые показатели деятельности, привязанные к основным бизнес-процессам компании;  
   • системно перейти к более детальным описаниям.

##### Рис. 2.3.2. Как описать бизнес-процессы?

   Действительно, корневая модель БП содержит в себе зародыши различных детальных описаний порядка исполнения процессов – описание бизнес-процессов в виде функций (функциональное описание), в виде алгоритмов, блок-схем, потоков объектов, которые образуются в компании в ходе ее функционирования (потоки материальных ресурсов, информационных ресурсов и финансовых ресурсов).

**Полезный для расширения восприятия комментарий.** Финансовые потоки и их модели используются при «финансовой оцифровке» процессов, при построении операционных бюджетов компании. Система операционных бюджетов компании представляет собой оцифрованные основные процессы деятельности, а свод операционных бюджетов в финансовые (бюджет доходов и расходов (БДР), бюджет по балансовому листу (ББЛ), бюджет движения денежных средств (БДДС) позволяет строить стандартные финансовые отчеты (рис. 2.3.3). Поэтому бюджетирование часто начинается с построения корневой модели БП.

1. Гибкое проектирование
2. Экстремальное программирование