Вопросы и ответы на ПИ

Легенда:

Ответ уже есть в моих конспектах лекций :Р

Оптимизированный полный ответ

Большой, мало сокращенный, либо неполный ответ

Оглавление

[Специфические особенности ПС. ПС как новый вид товарной продукции. 4](#_Toc408426247)

[Жизненный цикл ПС. Содержание основных этапов жизненного цикла ПС. 5](#_Toc408426248)

[Анализ и разработка требований к ПС. 7](#_Toc408426249)

[Определение целей создания ПС. Разработка внешних спецификаций на ПС. 8](#_Toc408426250)

[Цели и порядок внутреннего проектирования ПС. 9](#_Toc408426251)

[Прогнозирование технико-экономических показателей проектов ПС. 10](#_Toc408426252)

[Модульная структура ПС. Внешнее проектирование модулей. 11](#_Toc408426253)

[Проектирование и кодирование модулей. 12](#_Toc408426254)

[Принципы и методы тестирования ПС. Проектирование теста. 14](#_Toc408426255)

[Альтернативный 9 вопрос, ограниченное издание: 17](#_Toc408426256)

[Общая характеристика методов тестирования. 19](#_Toc408426257)

[Ручные и машинные методы тестирования. 20](#_Toc408426258)

[Методы структурного и функционального тестирования 22](#_Toc408426259)

[Тестирование модулей. Тестирование комплексов программ. 23](#_Toc408426260)

[Отладка программ. 24](#_Toc408426261)

[Документирование ПС. Состав документации на ПС. 25](#_Toc408426262)

[Испытания и сертификация ПС. 28](#_Toc408426263)

[Методы, технология, средства обеспечения сертификации ПС. 31](#_Toc408426264)

[Сопровождение и конфигурационное управление ПС. 32](#_Toc408426265)

[Особенности современных методологий и технологий разработки ПС. 33](#_Toc408426266)

[Технология и стандарты структурного программирования. 34](#_Toc408426267)

[Технология сборочного программирования. 35](#_Toc408426268)

[Направления развития и модели концепции открытых систем. 36](#_Toc408426269)

[Показатели качества ПС. Стандарты, регламентирующие показатели качества ПС. 37](#_Toc408426270)

[Выбор и измерение показателей качества ПС. 39](#_Toc408426271)

[Методы определения численных значений показателей качества ПС. 41](#_Toc408426272)

[Применения метрик в управлении качеством ПС. 42](#_Toc408426273)

[Понятие сложности ПС. Основные компоненты сложности ПС. 43](#_Toc408426274)

[Показатели вычислительной сложности ПС. 44](#_Toc408426275)

[Измерение и оценка сложности ПС. 45](#_Toc408426276)

[Основные понятия и виды корректности программ. 46](#_Toc408426277)

[Типы эталонов, методы измерений и проверки корректности программ. 47](#_Toc408426278)

[Понятие ошибки в программе. Источники ошибок. Классификационная схема программных ошибок. 48](#_Toc408426279)

[Методы и средства тестирования и отладки программ. 50](#_Toc408426280)

[Определение надежности ПС. Показатели надежности ПС. 51](#_Toc408426281)

[Факторы, определяющие надежность ПС. 52](#_Toc408426282)

[Статические и динамические модели надежности ПС. 53](#_Toc408426283)

[Методы обеспечения технологической безопасности ПС и данных. 55](#_Toc408426284)

[Эмпирические модели надежности ПС. 56](#_Toc408426285)

[Методы сбора и обработки данных о разработках ПС. 57](#_Toc408426286)

[Задачи и проблемы сертификации ПС. 58](#_Toc408426287)

[Виды сертификационных испытаний и программ. 60](#_Toc408426288)

[Методы, технология, средства обеспечения сертификации программных средств. 61](#_Toc408426289)

[Особенности современных программных средств и баз данных как объектов разработки. 62](#_Toc408426290)

[Цели и принципы документирования программных средств. Стандарты документирования программных средств. 63](#_Toc408426291)

[Технологическая и эксплуатационная документация на ПС. 64](#_Toc408426292)

[Организация документирования программных средств. Управление документированием этапов жизненного цикла ПС. 65](#_Toc408426293)

[Документация управления качеством ПС. 67](#_Toc408426294)

[Структура и содержание документов по этапам жизненного цикла ПС. 68](#_Toc408426295)

[Состав пользовательской документации на ПС. 71](#_Toc408426296)

[Техническое задание на проектирование ПС. Эскизный (технический), рабочий проект ПС. 72](#_Toc408426297)

[Документация тестирования компонентов и комплексов программ. Документация испытаний комплексов программ. 73](#_Toc408426298)

[Документация сопровождения и конфигурационного управления версиями программ. 74](#_Toc408426299)

[Общая характеристика инструментальных средств документирования. Пакеты программ для формирования отчетов. 75](#_Toc408426300)

[Пакеты конфигурационного управления. 76](#_Toc408426301)

# Специфические особенности ПС. ПС как новый вид товарной продукции.

Разработка программных средств имеет ряд специфических особенностей:  
• Прежде всего, следует отметить некоторое противостояние: неформальный характер требований к ПС (постановки задачи), но формализованный основной объект разработки ? программы ПС. Тем самым разработка ПС содержит определенные этапы формализации, а переход от неформального к формальному существенно неформален.  
• Разработка ПС носит творческий характер (на каждом шаге приходится делать какой-либо выбор, принимать какое-либо решение), а не сводится к выполнению какой-либо последовательности регламентированных действий. Тем самым эта разработка ближе к процессу проектирования каких-либо сложных устройств, но никак не к их массовому производству. Этот творческий характер разработки ПС сохраняется до самого ее конца.  
• Следует отметить также особенность продукта разработки. Он представляет собой некоторую совокупность текстов (т.е. статических объектов), смысл же (семантика) этих текстов выражается процессами обработки данных и действиями пользователей, запускающих эти процессы (т.е. является динамическим). Это предопределяет выбор разработчиком ряда специфичных приемов, методов и средств.  
• Продукт разработки имеет и другую специфическую особенность: ПС при своем использовании (эксплуатации) не расходуется и не расходует используемых ресурсов.

[В оглавление](#_top)

# Жизненный цикл ПС. Содержание основных этапов жизненного цикла ПС.

Под жизненным циклом ПС (software life cycle) понимают весь период его разработки и эксплуатации (использования), начиная от момента возникновения замысла ПС и кончая прекращением всех видов его использования. Жизненный цикл охватывает довольно сложный процесс создания и использования ПС (software process).  
В настоящее время можно выделить 5 основных подходов к организации процесса создания и использования ПС.  
• Водопадный подход. При таком подходе разработка ПС состоит из цепочки этапов. На каждом этапе создаются документы, используемые на последующем этапе. В исходном документе фиксируются требования к ПС. В конце этой цепочки создаются программы, включаемые в ПС.  
• Исследовательское программирование. Этот подход предполагает быструю (насколько это возможно) реализацию рабочих версий программ ПС, выполняющих лишь в первом приближении требуемые функции. После экспериментального применения реализованных программ производится их модификация с целью сделать их более полезными для пользователей. Этот процесс повторяется до тех пор, пока ПС не будет достаточно приемлемо для пользователей. Такой подход применялся на ранних этапах развития программирования, когда технологии программирования не придавали большого значения (использовалась интуитивная технология). В настоящее время этот подход применяется для разработки таких ПС, для которых пользователи не могут точно сформулировать требования (например, для разработки систем искусственного интеллекта).  
• Прототипирование. Этот подход моделирует начальную фазу исследовательского программирования вплоть до создания рабочих версий программ, предназначенных для проведения экспериментов с целью установить требования к ПС. В дальнейшем должна последовать разработка ПС по установленным требованиям в рамках какого-либо другого подхода (например, водопадного).  
• Формальные преобразования. Этот подход включает разработку формальных спецификаций ПС и превращение их в программы путем корректных преобразований. На этом подходе базируется компьютерная технология (CASE-технология) разработки ПС.  
• Сборочное программирование. Этот подход предполагает, что ПС конструируется, главным образом, из компонент, которые уже существуют. Должно быть некоторое хранилище (библиотека) таких компонент, каждая из которых может многократно использоваться в разных ПС. Такие компоненты называются повторно используемыми (reusable). Процесс разработки ПС при данном подходе состоит скорее из сборки программ из компонент, чем из их программирования.  
В рамках водопадного подхода различают следующие стадии жизненного цикла ПС: разработку ПС, производство программных изделий (ПИ) и эксплуатацию ПС.

1. Стадия разработки (development) ПС состоит из этапа его внешнего описания, этапа конструирования ПС, этапа кодирования (программирование в узком смысле) ПС и этапа аттестации ПС. Всем этим этапам сопутствуют процессы документирования и управления (management) ПС. Этапы конструирования и кодирования часто перекрываются, иногда довольно сильно. Это означает, что кодирование некоторых частей программного средства может быть начато до завершения этапа конструирования.  
   — Этап внешнего описания ПС – создание документа, являющегося описанием поведения ПС с точки зрения внешнего по отношению к нему наблюдателя с фиксацией требований относительно его качества. Внешнее описание ПС начинается с анализа и определения требований к ПС со стороны пользователей (заказчика), а также включает процессы спецификации этих требований.  
   — Конструирование (design) ПС охватывает процессы: разработку архитектуры ПС, разработку структур программ ПС и их детальную спецификацию.  
   — Кодирование (coding) ПС включает процессы создания текстов программ на языках программирование, их отладку с тестированием ПС.  
   — На этапе аттестации (acceptance) ПС производится оценка качества ПС. Если эта оценка оказывается приемлемой для практического использования ПС, то разработка ПС считается законченной. Это обычно оформляется в виде некоторого документа, фиксирующего решение комиссии, проводящей аттестацию ПС.  
   2. Программное изделие (ПИ)- экземпляр или копия разработанного ПС. Изготовление ПИ - это процесс генерации и/или воспроизведения (снятия копии) программ и программных документов ПС с целью их поставки пользователю для применения по назначению. Производство ПИ - это совокупность работ по обеспечению изготовления требуемого количества ПИ в установленные сроки. Стадия производства ПИ в жизненном цикле ПС является, по-существу, вырожденной (не существенной), так как представляет рутинную работу, которая может быть выполнена автоматически и без ошибок. Этим она принципиально отличается от стадии производства различной техники. В связи с этим в литературе эту стадию, как правило, не включают в жизненный цикл ПС.  
   3. Стадия эксплуатации ПС охватывает процессы хранения, внедрения и сопровождения ПС, а также транспортировки и применения ПИ по своему назначению. Она состоит из двух параллельно проходящих фаз: фазы применения ПС и фазы сопровождения ПС.  
   — Применение (operation) ПС - использование ПС для решения практических задач на компьютере путем выполнения ее программ.  
   — Сопровождение (maintenance) ПС - процесс сбора информации качестве ПС в эксплуатации, устранения обнаруженных в нем ошибок, его доработки и модификации, а также извещения пользователей о внесенных в него изменениях.

[В оглавление](#_top)

# Анализ и разработка требований к ПС.

[В оглавление](#_top)

# Определение целей создания ПС. Разработка внешних спецификаций на ПС.

Цель разработки программной системы - получить инструментарий, позволяющий строить когнитивные модели социально-экономических систем, проверять их адекватность, проводить анализ динамических и структурных свойств построенных моделей, прогнозировать развитие ситуаций и формировать программы управляющих воздействий.

Внешнее проектирование - это процесс описания планируемого поведения  
разрабатываемого ПИ с точки зрения потенциальных пользователей. Целью  
этого процесса является конкретизация внешних взаимодействий будущего  
ПИ без детализации внутреннего устройства.  
Внешний проект представляет собой внешние спецификации ПИ, предназначенные для различных групп специалистов пользователей и разработчиков.

Обычно внешние спецификации представляют собой объемистый документ. Поэтому для упрощения процесса его разработки применяют иерархическую организацию. Разработка внешних спецификаций разбивается на две части:  
предварительный внешний проект;  
детальный внешний проект.  
Предварительный внешний проект содержит описание основных компонентов, затем компонентов, из которых состоят эти основные компоненты, и далее внешних функций (функций пользователя), составляющих отдельные компоненты проекта. Причем предварительный внешний проект содержит все функции пользователя, но их точный синтаксис, семантика, выходные результаты остаются неопределенными.   
Детальный внешний проект каждой функции пользователя должен включать следующую информацию:  
1.  
Описание входных данных.  
Точное описание синтаксиса (формат, допустимые значения, области изменения) и семантики всех данных, вводимых пользователем.  
2.  
Описание выходных данных.  
Точное описание всех результатов функции (ответы на запрос терминала, сообщения об ошибках, контрольные сигналы, отчеты). Описание функциональной связи между входными и выходными данными, чтобы специалист, читающий спецификации, мог представить себе выходные данные, получаемые из любой комбинации входных данных, как правильной, так и неверной

3.Преобразование системы.  
Многие внешние функции не только порождают выходные данные, но и изменяют состояние системы. Так как рассматриваются внешние спецификации, эти изменения должны быть описаны с точки зрения пользователя.  
4.  
Характеристики надежности.  
Здесь описывается влияние всех возможных отказов функций на систему, файлы пользователя.  
5.  
Эффективность.  
Описание всех ограничений, которые накладываются на эффективность функции (например, затрачиваемое время, занимаемая память).  
  
[В оглавление](#_top)

# Цели и порядок внутреннего проектирования ПС.

**Проектирование программного обеспечения** — процесс создания проекта программного обеспечения (ПО), а также дисциплина, изучающая методы проектирования. Проектирование ПО является частным случаем Проектирования продуктов и процессов.

Целью проектирования является определение внутренних свойств системы и детализации её внешних (видимых) свойств на основе выданных заказчиком [требований к ПО](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B5%D0%B1%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%BC%D1%83_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8E) (исходные условия задачи). Эти требования подвергаются анализу.  
Первоначально программа рассматривается как чёрный ящик. Ход процесса проектирования и его результаты зависят не только от состава требований, но и выбранной модели процесса, опыта проектировщика.

[Модель](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C) предметной области накладывает ограничения на [бизнес-логику](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B7%D0%BD%D0%B5%D1%81-%D0%BB%D0%BE%D0%B3%D0%B8%D0%BA%D0%B0) и [структуры данных](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85).

В зависимости от класса создаваемого ПО, процесс проектирования может обеспечиваться как «ручным» проектированием, так и различными средствами его автоматизации. В процессе проектирования ПО для выражения его характеристик используются различные нотации — [блок-схемы](https://ru.wikipedia.org/wiki/%D0%91%D0%BB%D0%BE%D0%BA-%D1%81%D1%85%D0%B5%D0%BC%D0%B0), [ER-диаграммы](https://ru.wikipedia.org/wiki/ER-%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), [UML](https://ru.wikipedia.org/wiki/UML)-диаграммы, [DFD](https://ru.wikipedia.org/wiki/DFD)-диаграммы, а также макеты.

Проектированию обычно подлежат:

* [Архитектура ПО](https://ru.wikipedia.org/wiki/%D0%90%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%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);
* Устройство компонентов ПО;
* [Пользовательские интерфейсы](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F).

В российской практике проектирование ведется поэтапно в соответствии со стадиями, регламентированными ГОСТ 2.103-68: Техническое задание, Техническое предложение, Эскизный проект, Технический проект, Рабочий проект. На каждом из этапов формируется свой комплект документов, называемый проектом (проектной документацией).  
В зарубежной практике регламентирующими документами, например, являются Software Architecture Document, Software Design Document.

[В оглавление](#_top)

# Прогнозирование технико-экономических показателей проектов ПС.

Технология прогнозирования технико-экономических показателей и планирования разработки ПС предусматривает три фазы в соответствии с поэтапным уточнением исходных характеристик объекта и среды разработки в процессе проектирования:

* первичную оценку трудоемкости, длительности разработки и числа необходимых специалистов при учете минимума факторов — только предполагаемого объема и класса ПС при подготовке технического задания на проект ПС;
* расчет значений трудоемкости, длительности разработки и числа необходимых специалистов, а также представление типового первичного плана проведения работ с учетом 15 основных факторов, учитывающих характеристики ПС и среды разработки, полученные в процессе предварительного (эскизного) проектирования;
* корректировку типового плана проведения работ с учетом конкретных особенностей среды разработки и характеристик ПС и формирование рабочего плана проведения работ на этапах предварительного (эскизного) и детального (технического) проектирования, выдачу рекомендаций о составе отчетных документов на этапах процесса проектирования.

Фаза 1. При подготовке концепции проекта ПС определяются назначение и класс создаваемых программ. Сопоставлением с прототипами и аналогичными разработками может быть приближенно оценен объем создаваемого ПС. Исходной является отечественная и зарубежная статистика реальных проектов и аппроксимация основных зависимостей Эти данные позволяют качественно оценить риск проекта и целесообразность продолжения разработки спецификаций требований и заключения контракта.

Фаза 2. При завершении подготовки спецификации требований уточняются функции и основные компоненты ПС, что позволяет повысить достоверность определения объема и характеристик создаваемого комплекса. Для этого этапа возможна оценка трудоемкости, длительности и числа необходимых специалистов.

Фаза 3. В процессе подготовки предварительного (эскизного) проекта определяются: архитектура комплекса программ, распределение вычислительных ресурсов на его реализацию, функции и характеристики основных компонент, а также технология и средства автоматизации разработки. Тем самым объект и среда разработки уточняются до такого уровня, что становится рентабельным детальное планирование последующего проектирования.

[В оглавление](#_top)

# Модульная структура ПС. Внешнее проектирование модулей.

Много инфы тут: [vit-prog.narod.ru/page/TRPP/section\_1/subject\_1.3.htm](http://vit-prog.narod.ru/page/TRPP/section_1/subject_1.3.htm)

МОДУЛЬНОЕ ПРОГРАММИРОВАНИЕ (modular programming). Способ разработки программ, при котором программа разбивается на относительно независимые составные части - программные модули. При этом каждый модуль может разрабатываться, программироваться, транслироваться и тестироваться независимо от других. Внутреннее строение модуля для функционирования всей программы, как правило, значения не имеет. При модификации алгоритма, реализуемого модулем, структура программы не должна меняться.

Перечень основных характеристик программного модуля:

1. Размер модуля;
2. Прочность модуля;
3. Сцепление с другими модулями;
4. Рутинность модуля (независимость от предыстории обращений);

Два основных метода разработки структуры программы:

Метод восходящей разработки

1. Строится модульная структура программы в виде дерева;
2. Поочерёдно программируются модули программы, начиная с модулей самого нижнего уровня;
3. Производится поочерёдная отладка и тестирование модулей в том же порядке, как и их программирование.

Метод нисходящей разработки

1. Строится модульная структура программы в виде дерева;
2. Поочерёдно программируются модули программы, начиная с модулей самого верхнего (головного) уровня;

Производится поочерёдная отладка и тестирование модулей в том же порядке, как и их программирование. Несуществующие модули более низкого уровня заменяются имитаторами (заглушками).

[В оглавление](#_top)

# Проектирование и кодирование модулей.

Плюс можно использовать предыдущий билет для воды

Неотъемлемым условием разработки является планирование.   
Планирование кодирования и тестирования состоит в установлении очередности (порядка) кодирования и тестирования модулей разных уровней. Нужно выбирать такой порядок, который наиболее подходит для рассматриваемого проекта.

Возможны два "чистых" варианта, определяемых направлениями в иерархической схеме:

- иерархическая модель: сначала кодируются и тестируются все модули одного уровня, затем следующего уровня и т.д.  
- операционная модель: кодируются и тестируются все модули одной ветви иерархии, затем следующей ветви и т.д.

В каждой из этих моделей используется понятие "заглушка".  
Заглушкой называется некоторый простой модуль, подменяющий настоящий модуль на время тестирования вызывающего его модуля и имитирующий в какой-либо мере его исполнение (например, поставляющий некоторые фиксированные данные или просто сообщающий о своем исполнении).

При кодировании и тестировании модуля некоторого уровня используемые им модули более низкого уровня заменяются заглушками. По окончании тестирования этого модуля все заглушки заменяются настоящими модулями (которые поочередно кодируются и тестируются), а в виде заглушек представляются их преемники - модули следующего уровня.

В иерархической модели в начале кодирования и тестирования очередного уровня должны быть представлены в виде заглушек все модули этого уровня. Они поочередно заменяются настоящими модулями, одновременно в виде заглушек появляются все модули следующего уровня.

В операционной модели модули обычно разрабатываются в порядке их выполнения в готовой программе. Такой подход предпочтительнее, когда модули одного уровня связаны друг с другом по большому числу сложно структурированных данных. Нужно сначала создать и проверить модули (как правило, одной ветви), порождающие эти данные, а затем модули, использующие эти данные. Заглушки при этом создаются в порядке, определяемом кодируемыми и тестируемыми ветвями вычислений, а не целиком для всего уровня.

Как правило, в реальных проектах применяется комбинированный подход, сочетающий иерархическую и операционную модель. Выбирается такое "комбинированное" направление кодирования и тестирования, которое обеспечивает наиболее полное тестирование модулей и позволяет как можно раньше решить главные проблемы.

В качестве примера проекта, для кодирования и тестирования которого подходит иерархическая модель, можно указать программу тестирования различных методов сортировки массива. В головном модуле задаются тестовые массивы и вызываются модули второго уровня, выполняющие методы сортировки. Некоторые из них используют модули третьего уровня - индивидуальные для каждого из модулей второго уровня. Наибольшая сложность заключена в модулях второго уровня. Поскольку эти модули не зависят друг от друга ни по данным, ни по порядку их исполнения, то для кодирования и тестирования здесь естественно применить иерархическую модель.

Для программы, выполняющей ввод и обработку сводной экзаменационной ведомости и представленной выше иерархией модулей, естественно выбрать комбинацию операционной и иерархической моделей. Здесь наиболее важная часть - модуль первого уровня, содержащий описание структуры ведомости, задание операций над нею и вызов соответствующих модулей второго уровня. Далее в первую очередь должны быть реализованы модули (второго и третьего уровней), выполняющие ввод ведомости, так как без предварительного ее формирования невозможно тестирование никаких других операций.

[В оглавление](#_top)

# Принципы и методы тестирования ПС. Проектирование теста.

Про тесты есть [тут](http://www.osp.ru/os/2008/07/5478839/), но это скорее для личного пользования

[Полная версия](http://www.rae.ru/monographs/141-4632) представленного ниже материала

Тестирование программного средства (ПС) - это процесс выполнения программ на некотором наборе данных, для которого заранее известен результат применения или известны правила поведения этих программ. Указанный набор данных называется тестовым или просто тестом. Тестирование программ является одной из составных частей более общего понятия - «отладка программ». Под отладкой по­нимается процесс, позволяющий получить программу, функциони­рующую с требующимися характеристиками в заданной области изменения входных данных.

Проектирование теста:

Возможны следующие подходы к стратегии проектирования тестов:

1. Тестирование по отношению к спецификациям (не заботясь о тексте программы).

2. Тестирование по отношению к тексту программы (не заботясь о спецификациях).

Сторонники первого подхода проектирует тесты, исследуя внешние спецификации или спецификации сопряжения программы или модуля, которые он тестирует. Программа рассматривается как черный ящик. Логика их такова: «Меня не интересует, как выглядит эта программа, и выполнил ли я все команды. Я удовлетворен, если программа будет вести себя так, как указано в спецификациях». То есть в идеале проверить все возможные комбинации и значения на входе.

Сторонники второго подхода проектируют свои тесты, изучая логику программы. Они начинают с того, что стремятся подготовить достаточное число тестов, чтобы каждая команда была выполнена, хотя бы, один раз. Чтобы каждая команда условного перехода выполнялась в каждом направлении хотя бы раз. Их идеал - проверить каждый путь, каждую ветвь алгоритма. При этом не интересуются спецификациями.

Принципы тестирования:

Существуют несколько эмпирических правил проведения тес­тирования программ, обобщающих опыт тестировщиков.

1. Процесс тестирования более эффективен, если проводится не автором программы. По своей сути тестирование - это процесс деструктивный (разрушительный). Именно этим и объясняется, поче­му многие считают его трудным. Особенно трудным и малоэффек­тивным он является для самого автора программы, так как после выполнения конструктивной части при проектировании и написания программы, ему трудно перестроиться на деструктивный образ мышления и, создав программу, тут же приступить к пристрастному выявлению в ней ошибок. Поэтому для проведения тестирования создаются специальные группы тестирования. Это не означает, что программист не может тестировать свою программу. Речь идет о повышении эффективности тестирования.

2. Необходимой частью тестового набора данных должно быть описание предполагаемых значений результатов тестовых прогонов. Тестирование как процесс многократного выполнения про­граммы проводится на многочисленных входных наборах данных. Чтобы определить правильность полученных в результате очеред­ного тестового прогона данных, необходимо знать ожидаемый ре­зультат. Таким образом, тестовый набор данных должен включать в себя два компонента: описание входных данных, описание точного и кор­ректного результата, соответствующего набору входных данных. Этот принцип сложно, а в некоторых случаях и невозможно реализовать на практике. Сложность его заключается в том, что при тестировании программы (модуля) необходимо для каждого входного набора данных рассчитать вручную ожидаемый результат или найти допустимый интервал изменения выходных данных. Процесс этот трудоемкий даже для небольших про­грамм, так как он требует ручных расчетов, следуя логике алгоритма программы. Из рассмотренного принципа, который трудно реализуем, но которого следует придерживаться логически, вытекает следующий.

3. Необходимо изучить результаты каждого теста. Из практики следует, что значительная часть обнаруженных ошибок могла быть выявлена в результате первых тестовых прогонов, но они были пропущены вследствие недостаточно тщательного анализа их результатов.

4. Тесты для неправильных и непредусмотренных входных данных должны разрабатываться также тщательно, как для правильных и предусмотренных. Согласно этому принципу при обработке данных, выходящих за область допустимых значений, в тестируемой программе должна быть предусмотрена диагностика в виде сообщений. Если сообще­ние о причине невозможности обработки по предложенному алго­ритму отсутствует, и программа завершается аварийно или ведет себя непредсказуемо, то такая программа не может считаться ра­ботоспособной и требует существенной доработки. Тестовые наборы данных из области недопустимых входных значений обладают большей обнаруживающей способностью, чем тесты, соответствующие корректным входным данным.

5. Необходимо проверять не только, делает ли программа то, для чего она предназначена, но и не делает ли она того, чего не должна делать. Это утверждение логически вытекает из предыдущего. Необ­ходимо любую программу проверить на нежелательные побочные эффекты.

6. Следует тщательнее проверять те участки программ, где обнаруживается больше ошибок. Утверждается, что вероятность наличия необнаруженных ошибок в какой-либо части программы пропорциональна числу ошибок, уже обнаруженных в этой части. Возможно, что те части программы, где при тестировании об­наружено большее число ошибок, либо были слабо проработаны с точки зрения системного анализа, либо разрабатывались програм­мистами более низкой квалификации.

Методы тестирования:

В практике тестирования используются следующие по­следовательно применяемые методы: статический, детерминиро­ванный, стохастический и в реальном масштабе времени.

*Статическое тестирование* - проводится без использова­ния ЭВМ путем просмотра текста программы после трансляции, проверки правил структурного построения программ и обработки данных. В качестве эталонов используются, во-первых, внутренние спецификации, а, во-вторых, коллективный опыт специалистов-тестировщиков. Применение статического тестирования достаточ­но эффективно. Для типичных программ, по данным фирмы IBM, можно находить от 30 % до 80 % ошибок логического проектирования и кодирования. Этот метод способствует существенному повыше­нию производительности и надежности программ, позволяет раньше обнаружить ошибки, а значит уменьшить стоимость ис­правления.

*Детерминированное тестирование* - это многократное выполнение программы на ЭВМ с использованием определенных, специальным образом подобранных тестовых наборов данных. При детерминированном тестировании контролируются каждая комбинация исходных данных и соответствующие результаты, а также каждое утверждение в спецификации тестируемой програм­мы. Этот метод наиболее трудоемкий, поэтому детерминирован­ное тестирование применяется для отдельных модулей в процессе сборки программы или для небольших и несложных программных комплексов.

*Стохастическое тестирование* предполагает использова­ние в качестве исходных данных множество случайных величин с соответствующими распределениями. Для сравнения полученных результатов используются также распределения случайных вели­чин. Стохастическое тестирование применяется в основном для обнаружения ошибок, а для диагностики и локализации ошибок приходится переходить к детерминированному тестированию с использованием конкретных значений исходных данных, из облас­ти изменения ранее использовавшихся случайных величин. Сто­хастическое тестирование наилучшим образом подвергается ав­томатизации путем использования датчиков случайных чисел (генераторов случайных величин) и применяется для комплексно­го тестирования ППП. Паке́т прикладны́х програ́мм

*Тестирование в реальном масштабе времени* осуществ­ляется для ППП, предназначенных для работы в системах реаль­ного времени. В процессе такого тестирования проверяются ре­зультаты обработки исходных данных с учетом времени их поступления, длительности и приоритетности обработки, динамики использования памяти и взаимодействия с другими программами. При обнаружении отклонения результатов выполнения программ от ожидаемых для локализации ошибок, приходится фиксировать время и переходить к детерминированному тестированию.

[В оглавление](#_top)

## Альтернативный 9 вопрос, ограниченное издание:

**Принципы тестирования**

Тестирование проводится для того, чтобы найти немногие оставшиеся ошибки в хорошо спроектированной системе и тем самым повысить ее надежность, а следовательно, ценность. С помощью тестирования нельзя добиться хорошей надежности в плохо спроектированной системе.

Удачным считается тест, который обнаружил ошибку.   
Если ни одна ошибка не была обнаружена, то тест считается неудачным.

* Ошибки имеют свойство группироваться.   
  Если в какой-то части программы найдено много ошибок, то там еще много осталось.
* Никогда не изменяйте программу, чтобы облегчить ее тестирование.
* Следует избегать тестирования программы ее автором. Если программист сделал ошибку при написании программы, то вполне вероятно он сделает ту же самую ошибку при ее тестировании. Программист подсознательно считает свою программу продолжением самого себя и не станет особенно тщательно ее тестировать.
* Разработка тестов -- творческий процесс, который требует, в некотором роде, разрушительного склада ума.
* Хорош тот тест, для которого высока вероятность обнаружить ошибку.
* Необходимо проверять не только, делает ли программа то, для чего она предназначена, но и не делает ли она то, что не должна делать.
* Некоторую часть тестов следует выделить в качестве тестов регрессии, которые в будущем должны выполняться после каждого исправления программы, чтобы проверить, не ухудшилась ли система, не произошел ли регресс.
* Недостаток комментариев усложняет поиск ошибок, так как при проверке бывает трудно разобраться в сложной программе без небольших пояснений.
* Избыток комментариев также усложняет поиск ошибок. Комментарии говорят, что делает программа по мнению автора, а не что она делает на самом деле.

Восходящее и Нисходящее тестирование.

При восходящем подходе программа собирается и тестируется снизу вверх. Только модули самого нижнего уровня («терминальные» модули; модули, не вызывающие других модулей) тестируются изолированно, автономно. После того как тестирование этих модулей завершено, вызов их должен быть так же надежен, как вызов встроенной функции языка или оператор присваивания. Затем тестируются модули, непосредственно вызывающие уже проверенные. Эти модули более высокого уровня тестируются не автономно, а вместе с уже проверенными модулями более низкого уровня. Процесс повторяется до тех пор, пока не будет достигнута вершина. Здесь завершаются и тестирование модулей, и тестирование сопряжений программы.

Достоинства:

1. Простота подготовки тестов.

2. Возможность полной реализации плана тестирования модуля.

Недостатки:

1. Тестовые данные готовятся, как правило, не в той форме, которая рассчитана на пользователя.

2. Большой объём отладочного программирования.

3. Необходимость специального тестирования сопряжения модулей.

При нисходящем подходе программа собирается и тестируется сверху вниз. Изолировано тестируется только головной модуль. После того как тестирование этого модуля завершено, с ним соединяются (например, редактором связей) один за другим модули, непосредственно вызываемые им, и тестируется полученная комбинация. Процесс повторяется до тех пор, пока не будут собраны и проверены все модули.

Достоинства:

1. Большинство тестов готовится в форме, рассчитанной на пользователя.

2. Во многих случаях относительно небольшой объём объём отладочного программирования.

3. Отпадает необходимость тестирования сопряжения модулей.

Недостатки:

1.Тестовое состояние информационной среды готовится косвенно – как результат применения уже отлаженных модулей к тестовым данным или данным, выдаваемым имитаторами.

Однако еще методы есть в [оригинале](#_Принципы_и_методы), а так же [12](#_Методы_структурного_и) и [13](#_Тестирование_модулей._Тестирование) вопросе

[В оглавление](#_top)

# Общая характеристика методов тестирования.

Ваще хз, коси под дурачка и рассказывай [прошлый вопрос](#_Принципы_и_методы)

[В оглавление](#_top)

# Ручные и машинные методы тестирования.

Автоматизированное тестирование программного обеспечения — часть процесса тестирования на этапе контроля качества в процессе разработки программного обеспечения. Оно использует программные средства для выполнения тестов и проверки результатов выполнения, что помогает сократить время тестирования и упростить его процесс.

Наиболее распространенной формой автоматизации является тестирование приложений через графический пользовательский интерфейс (англ. *GUI*).

GUI-автоматизация развивалась в течение 4 поколений инструментов и техник:

* Утилиты записи и воспроизведения (англ. *capture/playback tools*) записывают действия тестировщика во время ручного тестирования. Они позволяют выполнять тесты без прямого участия человека в течение продолжительного времени, значительно увеличивая продуктивность и устраняя «тупое» повторение однообразных действий во время ручного тестирования. В то же время, любое малое изменение тестируемого ПО требует перезаписи ручных тестов. Поэтому это первое поколение инструментов не эффективно и не масштабируемо.
* Написание сценария (англ. *scripting*) — форма программирования на языках, специально разработанных для автоматизации тестирования ПО — смягчает многие проблемы инструментов записи и воспроизведения. Но разработкой занимаются программисты высокого уровня, которые работают отдельно от тестировщиков, непосредственно запускающих тесты. К тому же скрипты более всего подходят для тестирования GUI и не могут быть внедренными, пакетными или вообще каким-либо образом объединены в систему. Наконец, изменения в тестируемом ПО требуют сложных изменений в соответствующих скриптах, и поддержка все возрастающей библиотеки тестирующих скриптов становится в конце концов непреодолимой задачей.
* Управляемое данными тестирование (англ. *Data-driven testing*) — методология, которая используется в автоматизации тестирования. Особенностью является то, что тестовые скрипты выполняются и верифицируются на основе данных, которые хранятся в центральном хранилище данных или базе данных. Роль базы данных могут выполнять ODBC-ресурсы, csv или xls файлы и т. д. Управляемое данными тестирование — это объединение нескольких взаимодействующих тестовых скриптов и их источников данных в фреймворк, используемый в методологии. В этом фреймворке переменные используются как для входных значений, так и для выходных проверочных значений: в тестовом скрипте обычно закодированы навигация по приложению, чтение источников данных, ведение логов тестирования. Таким образом, логика, которая будет выполнена в скрипте, также зависит от данных.
* Тестирование по ключевым словам (англ. *Keyword-based*) автоматизация подразумевает разделение процесса создания кейсов на 2 этапа: этап планирования и этап реализации. В этом случае конечный тест представляет собой не программный код, а описание последовательности действий с их параметрами (например, «завести в базе данных пользователя с логином XXX и паролем YYY»). При этом фреймворк отвечает за непосредственную реализацию ключевых слов (действий), а дизайнеру тестов достаточно иметь представление о всём наборе действий, реализованных во фреймворке. Это даёт возможность создавать тесты людям, не имеющим навыков программирования.

Средства тестирования представляют собой программы, которые помогают автоматизировать процесс испытаний системы; их часто называют средствами автоматического тестирования. Наличие таких средств дает возможность небольшой группе программистов провести анализ программ большого объема, что в противном случае было бы неосуществимо.

Первым действенным средством автоматического тестирования является генератор тестовых данных (ГТД), который формирует данные для использования в проверяемой программе. Такие данные часто оказываются полезными как первый тест программы, позволяющий определить, пригодны ли конкретный модуль или подпрограмма к стыковке с другими элементами.

*Компаратор файлов* представляет собой программу, которая считывает два файла и выводит на печать их различающиеся элементы. Он может использоваться для сопоставления результатов, выданных при двух различных тестовых прогонах, с целью выявления каких-либо расхождений. При этом программа сравнения файлов распечатывает только обнаруженные различия, и ручная проверка требуется только для несовпадающих элементов.

Проверка результатов тестирования может осуществляться самой тестируемой программой. Этот процесс часто называют *самопроверкой* программного обеспечения.

[В оглавление](#_top)

# Методы структурного и функционального тестирования

* структурное тестирование (белый ящик),
* функциональное тестирование (черный ящик)

*Структурное тестирование*

При данном подходе считается, что текст программы виден (белый ящик).   
Тестируются блоки ветвлений, циклы и т.д.

Существует несколько типов структурного тестирования:

* покрытие операторов,
* покрытие решений,
* покрытие решений / условий,
* комбинаторное покрытие условий,
* тестирование циклов.

*Функциональное тестирование*

При данном подходе считается, что текст программы не виден, и программа рассматривается как черный ящик, т.е. известны входные и выходные условия, а также общая схема работы. Программа проверятся по ее спецификациям.

Существуют несколько видов функционального тестирования:

* эквивалентные классы,
* анализ граничных значений,
* тестирование на предельных нагрузках,
* тестирование на предельных объемах,
* тестирование защиты,
* эксплуатация системы самим разработчиком (если возможно),
* опытная эксплуатация.

[В оглавление](#_top)

# Тестирование модулей. Тестирование комплексов программ.

[Смотри Восходящее и Нисходящее тестирование.](#_Альтернативный_9_вопрос,)

Модифицированный нисходящий метод

Применяя нисходящее тестирование, часто невозможно тестировать определенные логические условия, например ошибочные ситуации или защитные проверки. Нисходящий метод, кроме того, делает сложной или вообще невозможной проверку исключительных ситуаций в некотором модуле, если программа работает с ним лишь в ограниченном контексте (это означает, что модуль никогда не получит достаточно полный набор входных значений). Даже если тестирование такой ситуации в принципе осуществимо, часто бывает трудно определить, какие именно нужны тесты, если они вводятся в точке программы, удаленной от места проверки соответствующего условия.

Подход, называемый модифицированным нисходящим методом, решает эти проблемы: требуется, чтобы каждый модуль прошел автономное тестирование перед подключением к программе. Это решает перечисленные проблемы, но здесь требуются и драйверы, и заглушки для каждого модуля.

Метод большого скачка

Один из подходов к интеграции модулей - это метод большого скачка. В соответствии с этим методом каждый модуль тестируется автономно. По окончании тестирования модулей они интегрируются в систему все сразу.

Метод сэндвича

При использовании этого метода одновременно начинают восходящее и нисходящее тестирование, собирая программу как снизу, так и сверху и встречаясь, где-то в середине. Например, если разработчик может представить свою систему в виде уровня прикладных модулей, затем уровня модулей обработки запросов, затем уровня примитивных функций, то он может решить применять нисходящий метод на уровне прикладных модулей (программируя заглушки вместо модулей обработки запросов), а на остальных уровнях применить восходящий метод.

Модифицированный метод сандвича

При тестировании методом сандвича возникает та же проблема, что и при нисходящем подходе. Проблема эта заключается в том, что невозможно досконально тестировать отдельные модули. Восходящий этап тестирования по методу сандвича решает эту проблему для модулей нижних уровней, но она может по-прежнему оставаться открытой для нижней половины верхней части программы. В модифицированном методе сандвича нижние уровни также тестируются строго снизу вверх. А модули верхних уровней сначала тестируются изолированно, а затем собираются нисходящим методом.

Таким образом, модифицированный метод сандвича тоже представляет собой компромисс между восходящим и нисходящим подходами.

[В оглавление](#_top)

# Отладка программ.

Отладка программного средства (ПС) - это деятельность, направленная на обнаружение и исправление ошибок в ПС с использованием процессов выполнения его программ. Отладку можно представить в виде многократного повторения трех процессов: тестирования, в результате которого может быть констатировано наличие в ПС ошибки, поиска места ошибки в программах и документации ПС и редактирования программ и документации с целью устранения обнаруженной ошибки.

Другими словами: Отладка = Тестирование + Поиск ошибок + Редактирование.

Иногда тестирование и отладку считают синонимами.

Бонус:

При исправлении ошибки высока вероятность внесения новой ошибки (примерно 20%).   
Если программу исправляет не автор, тогда вероятность еще выше.

* Изучите программу в окрестности найденной ошибки в поисках новых неприятностей, так как ошибки имеют свойство появляться группами. Вспомните похожие места в системе, где возможно была сделана такая же ошибка.
* Если ошибка была обнаружена при эксплуатации системы, то часто требуется устранить *последствия* ошибки. Здесь главное не усугубить положение поспешными и непродуманными действиями. Рекомендуется по возможности сделать резервную копию.
* Часто пользователи сами предлагают способы решения проблемы. Такие пути в будущем могут привести к еще более сложным проблемам. Все предложения надо критически проанализировать.
* Не все ошибки являются ошибками разработчиков, некоторые ошибки происходят из-за неправильных входных данных или действий пользователей. В таком случае стоит принять меры для недопущения таких ошибок в будущем

И пошел 9 вопрос, если этого не хватит

[В оглавление](#_top)

# Документирование ПС. Состав документации на ПС.

Документация на программное обеспечение — это документы, сопровождающие некоторое программное обеспечение (ПО) — программу или программный продукт. Эти документы описывают то, как работает программа и/или то, как её использовать.

Документирование — это процесс создания документации, важная часть в разработке программного обеспечения, ей часто уделяется недостаточно внимания.

Цель документирования – создание средства передачи информации между разработчиками ПС, средства управления разработкой ПС и средства передачи пользователям информации, необходимой для применения и сопровождения ПС.

Документацию можно разбить на две группы:

* Документы управления разработкой ПС.
* Документы, входящие в состав ПС.

*Документы управления разработкой ПС* (*software process documentation*) управляют и протоколируют процессы разработки и сопровождения ПС, обеспечивая связи внутри коллектива разработчиков ПС и между коллективом разработчиков и *менеджерами ПС* (*software managers*) - лицами, управляющими разработкой ПС. Эти документы могут быть следующих типов:

* *Планы, оценки, расписания.* Эти документы создаются менеджерами для прогнозирования и управления процессами разработки и сопровождения ПС.
* *Отчеты об использовании ресурсов в процессе разработки.* Создаютсяменеджерами.
* *Стандарты.* Эти документы предписывают разработчикам, каким принципам, правилам, соглашениям они должны следовать в процессе разработки ПС. Эти стандарты могут быть как международными или национальными, так и специально созданными для организации, в которой ведется разработка ПС.
* *Рабочие документы.* Это основные технические документы, обеспечивающие связь между разработчиками. Они содержат фиксацию идей и проблем, возникающих в процессе разработки, описание используемых стратегий и подходов, а также рабочие (временные) версии документов, которые должны войти в ПС.
* *Заметки и переписка.* Эти документы фиксируют различные детали взаимодействия между менеджерами и разработчиками.

*Документы, входящие в состав ПС* (*software product documentation*), описывают программы ПС как с точки зрения их применения пользователями, так и с точки зрения их разработчиков и сопроводителей (в соответствии с назначением ПС). Эти документы образуют два комплекта с разным назначением:

* Пользовательская документация ПС (П-документация).
* Документация по сопровождению ПС (С-документация).

Пользовательская документация ПС(*user documentation*) объясняет пользователям, как они должны действовать, чтобы применить разрабатываемое ПС.

Можно считать типичным следующий состав пользовательской документации для достаточно больших ПС:

* *Общее функциональное описание ПС*. Дает краткую характеристику функциональных возможностей ПС. Предназначено для пользователей, которые должны решить, насколько необходимо им данное ПС.
* *Руководство по инсталяции ПС*. Предназначено для администраторов ПС. Оно должно детально предписывать, как устанавливать системы в конкретной среде, в частности, должно содержать описание компьютерно-считываемого носителя, на котором поставляется ПС, файлы, представляющие ПС, и требования к минимальной конфигурации аппаратуры.
* *Инструкция по применению ПС.* Предназначена для ординарных пользователей. Содержит необходимую информацию по применению ПС, организованную в форме удобной для ее изучения.
* *Справочник по применению ПС*. Предназначен для ординарных пользователей. Содержит необходимую информацию по применению ПС, организованную в форме удобной для избирательного поиска отдельных деталей.
* *Руководство по управлению ПС*. Предназначено для администраторов ПС. Оно должно описывать сообщения, генерируемые, когда ПС взаимодействует с другими системами, и как должен реагировать администратор на эти сообщения. Кроме того, если ПС использует системную аппаратуру, этот документ может объяснять, как сопровождать эту аппаратуру.

*Документация по сопровождению ПС* (*system documentation*) описывает ПС с точки зрения ее разработки. Эта документация необходима, если ПС предполагает изучение того, как оно устроена (сконструирована), и модернизацию его программ.

Документация по сопровождению ПС можно разбить на две группы:

1. документация, определяющая строение программ и структур данных ПС и технологию их разработки;
2. документацию, помогающую вносить изменения в ПС.

Документация первой группы содержит итоговые документы каждого технологического этапа разработки ПС. Она включает следующие документы:

* Внешнее описание ПС (Requirements document).
* Описание архитектуры ПС (description of the system architecture), включая внешнюю спецификацию каждой ее программы (подсистемы).
* Для каждой программы ПС - описание ее модульной структуры, включая внешнюю спецификацию каждого включенного в нее модуля.
* Для каждого модуля - его спецификация и описание его строения (design description).
* Тексты модулей на выбранном языке программирования (program source code listings).
* Документы установления достоверности ПС (validation documents), описывающие, как устанавливалась достоверность каждой программы ПС и как информация об установлении достоверности связывалась с требованиями к ПС.

Документы установления достоверности ПС включают, прежде всего, документацию по тестированию (схема тестирования и описание комплекта тестов), но могут включать и результаты других видов проверки ПС, например, доказательства свойств программ. Для обеспечения приемлемого качества этой документации полезно следовать общепринятым рекомендациям и стандартам

Документация второй группы содержит *Руководство по сопровождению ПС* (system maintenance guide), которое описывает особенности реализации ПС (в частности, трудности, которые пришлось преодолевать) и как учтены возможности развития ПС в его строении (конструкции). В нем также фиксируются, какие части ПС являются аппаратно- и программно-зависимыми.

[В оглавление](#_top)

# Испытания и сертификация ПС.

В ряде случаев применение программного обеспечения может приводить к проявлению рисков, обусловленных как внутренними свойствами самих программных продуктов, так и возможностями внешнего воздействия на них. Все это приводит к необходимости проведения независимой, в том числе метрологической, экспертизы используемого программного обеспечения. Такая экспертиза может быть реализована, например, в виде сертификации программного обеспечения на предмет установления его соответствия требованиям соответствующей нормативной документации.

Для сертификации ПО необходимо наличие программного продукта и документации на него

Документация, подаваемая заявителем для прохождения процедуры добровольной сертификации программного обеспечения должна отображать следующую информация о ПО:

* официальное название ПО;
* описание структуры программного обеспечения, выполняемых им функций, в том числе последовательность обработки данных;
* описание метрологически значимых функций и параметров ПО, существенных для их работы (в соответствии с Рекомендацией МИ 2955-2010);
* описание реализованных в ПО вычислительных алгоритмов, а также их блок-схемы;
* описание модулей ПО;
* перечень интерфейсов и перечень команд для каждого интерфейса, в том числе для интерфейса связи и пользователя, включая заявление об их полноте;
* описание интерфейсов пользователя, всех меню и диалогов;
* список, значение и действие всех команд, получаемых от клавиатуры, мыши и других устройств ввода информации;
* описание реализованной методики идентификации ПО и самих идентификационных признаков;
* описание хранимых или передаваемых наборов данных;
* описание реализованных методов защиты ПО и данных;
* характеристики требуемых системных и аппаратных средств, если эта информация не приведена в руководстве пользователя.  
    
  Процесс сертификации

Процесс сертификации программного обеспечения включает в себя следующие этапы:

1. подача заказчиком заявки на сертификацию;

2. принятие решения по заявке на сертификацию, в том числе назначение экспертов на проведение основных работ по сертификации из числа экспертов органа по сертификации;

3. оформление договора на проведение работ по сертификации;

4. разработка методики проведения сертификационных испытаний ПС и согласование этой методики с заказчиком;

5. проведение сертификационных испытаний ПС;

6. принятие решения о выдаче Сертификата соответствия либо об отказе в выдаче Сертификата соответствия;

7. оформление Сертификата соответствия.

Сертификационные испытания ПС осуществляется в два этапа:

1. Технологические испытания. Проводятся с использованием современных методов и средств по формализованным правилам, удостоверяющим соответствие реальных количественных и качественных показателей тем, которые зафиксированы в НТД или программной документации;

2. Оценка, проводимая экспертами.

В ходе испытаний выполняется:

· Идентификация объекта испытаний путем проверки характеристик идентификации программного средства (полное название ПС, версия и дата выпуска ПС, сведения о разработчике ПС, сведения о входящих в состав компонентах, основные выполняемые функции, состав программной документации);

· Инсталляция путем установки программного продукта на компьютеры, на которые до этого данный программный продукт не был установлен;

· Экспертиза программной документации на соответствие требованиям Государственных стандартов ГОСТ Р ИСО/МЭК 12119-2000 (п. 3.2), ГОСТ Р ИСО 9127-94 (п.п. 5, 6.1, 6.3-6.5);

· Проверка и оценка качества сертифицируемого программного продукта в соответствии с требованиями нормативных документов (список документов определяется в процессе разработки методики), проверка программного продукта на соответствие выполняемых функций по руководству пользователя и требованиям технического задания.

Существуют два основных подхода к сертификации – и соответственно два типа нормативных документов.

Функциональное тестирование средств защиты информации, позволяющее убедиться в том, что продукт действительно реализует заявленные функции. Это тестирование чаще всего проводится на соответствие конкретному нормативному документу – например, одному из руководящих документов Гостехкомиссии России. Такие документы установлены, например, для межсетевых экранов и средств защиты от несанкционированного доступа. Если же не существует документа, которому сертифицируемый продукт соответствовал бы в полной мере, то функциональные требования могут быть сформулированы в явном виде – например, в технических условиях, или в виде задания по безопасности (в соответствии с положениями стандарта ГОСТ Р 15408).

Структурное тестирование программного кода на отсутствие недекларированных возможностей. Классическим примером недекларированных возможностей являются программные закладки, которые при возникновении определенных условий инициируют выполнение не описанных в документации функций, позволяющих осуществлять несанкционированные воздействия на информацию (по ГОСТ Р 51275-99). Выявление недекларированных возможностей предполагает проведение серии тестов исходных текстов программ, предоставление которых является необходимым условием для возможности проведения сертификационных испытаний.

[В оглавление](#_top)

# Методы, технология, средства обеспечения сертификации ПС.

Это все в [предыдущем](#_Испытания_и_сертификация) вопросе

[В оглавление](#_top)

# Сопровождение и конфигурационное управление ПС.

Конфигурационное управление (англ. software configuration management, SCM) в программной инженерии — комплекс методов, направленных на систематический учёт изменений, вносимых разработчиками в программный продукт в процессе его разработки и сопровождения, сохранение целостности системы после изменений, предотвращение нежелательных и непредсказуемых эффектов, формализацию процесса внесения изменений. В целом, конфигурационное управление отвечает на вопрос: «Кто-то уже сделал нечто, как нам это воспроизвести?»

К процедурам можно отнести создание резервных копий, контроль исходного кода, требований проекта, документации и т. д. Степень формальности выполнения данных процедур зависит от размеров проекта, и при правильной её оценке данная концепция может быть очень полезна.

*Цели конфигурационного управления:*

* Контроль: SCM позволяет отслеживать изменения в контролируемых объектах, обеспечивает соблюдение процесса разработки
* Управление: SCM диктует процесс автоматической идентификации в ходе всего жизненного цикла ПО, обеспечивает простоту модификации и сопровождения ПО
* Экономия средств: снижается риск потерь от ротации кадров в организации, предоставить возможность сменить организацию-разработчика без перепроектирования
* Качество

*Задачи конфигурационного управления:*

* идентификация конфигурации
* контроль конфигурации: контроль над изменениями материалов
* учёт текущего состояния: состояние документов, состояние кода, состояние отдельных задач и всего проекта в целом
* управление процессом разработки
* управление сборкой
* управление окружением
* отслеживание задач и проблем (в частности, отслеживание ошибок)

**Процедуры управления конфигурацией**

Ревизия конфигурации — процесс проверки того, что документ нижнего уровня соответствует всем требованиям документа верхнего уровня.

Аудит конфигурации — процесс проверки того, что готовый продукт или его часть соответствуют документации.

Контроль конфигурации — процесс, при котором все предлагаемые изменения продукта проходят одобрение специальной группы (или отдельного человека).

Учет состояния конфигурации — процесс подготовки отчетов о текущем состоянии продукта и состоянии утвержденных изменений.

[В оглавление](#_top)

# Особенности современных методологий и технологий разработки ПС.

Вот немного воды, надеюсь хватит

**Метод критического пути** — инструмент планирования расписания и управления сроками проекта.  
В основе метода лежит определение наиболее длительной последовательности задач от начала проекта до его окончания с учетом их взаимосвязи. Задачи, лежащие на критическом пути (*критические задачи*), имеют нулевой резерв времени выполнения, и, в случае изменения их длительности, изменяются сроки всего проекта. В связи с этим, при выполнении проекта критические задачи требуют более тщательного контроля, в частности, своевременного выявления проблем и рисков, влияющих на сроки их выполнения и, следовательно, на сроки выполнения проекта в целом.

Метод PERT представляется сетевыми диаграммами с вершинами событиями, а работа - в виде линии между двумя событиями, отображающими начало и конец работы. В целом расхождения между этими двумя методами сетевого представления графа работ - незначительные. Однако этот метод, в отличие от CMP, учитывает возникающие неопределенности во времени выполнения каждой операции.

Водопадная модель:

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

**Спира́льная модель**, предложенная Барри Боэмом в 1986 году, стала существенным прорывом в понимании природы разработки ПО. Она представляет собой процесс разработки программного обеспечения, сочетающий в себе как проектирование, так и постадийное прототипирование с целью сочетания преимуществ восходящей и нисходящей концепции, делающая упор на начальные этапы жизненного цикла: анализ и проектирование.

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.

[В оглавление](#_top)

# Технология и стандарты структурного программирования.

**Структу́рное программи́рование** — методология разработки программного обеспечения, в основе которой лежит представление программы в виде иерархической структуры блоков. Предложена в 1970-х годах Э. Дейкстрой и др.

Теорема Бёма-Якопини:

Любая программа, заданная в виде блок-схемы, может быть представлена с помощью трех управляющих структур:

* **последовательность** *— обозначается:* f THEN g,
* **ветвление** *— обозначается:* IF p THEN f ELSE g,
* **цикл** *— обозначается:* WHILE p DO f,

где f, g — блок-схемы с одним входом и одним выходом,

р — условие,

THEN, IF, ELSE, WHILE, DO — ключевые слова[[15]](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%BD%D0%BE%D0%B5_%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" \l "cite_note-15).

*Пояснение.* Формула f THEN g означает следующее: сначала выполняется программа f, затем выполняется программа g.

Принципы структурного программирования:

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

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

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

*Принцип 4.* Повторяющиеся фрагменты программы можно оформить в виде *подпрограмм* (процедур и функций).

*Принцип 5.* Каждую логически законченную группу инструкций следует оформить как блок. Блоки являются основой структурного программирования.

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

*Принцип 7.* Разработка программы ведётся пошагово, методом «сверху вниз»

[В оглавление](#_top)

# Технология сборочного программирования.

Сборочное программирование предполагает, что программа собирается путем переиспользования уже известных фрагментов.

Сборочное программирование решает задачу многократного и быстрого применения в процессе создания программы заранее изготовленных деталей.

Сборка может осуществляться вручную или быть задана на некотором языке сборки, или извлечена полуавтоматическим образом из специфики задачи. Основные направления для создания техники сборочного программирования:

-Выработка стиля программирования, соответствующего принятым принципам модульности.

-Повышение эффективности межмодульных интерфейсов; важность аппаратной поддержки модульности.

-Ведение большой базы программных модулей, решение проблемы идентификации модулей и проверки пригодности по описанию интерфейса. Модули должны стать “программными кирпичиками”, из которых строится программа.

[В оглавление](#_top)

# Направления развития и модели концепции открытых систем

Система, реализующая открытые спецификации на интерфейсы, сервисы и поддерживаемые форматы данных, достаточные для того, чтобы обеспечить должным образом разработанным приложениям возможность переноса с минимальными изменениями на широкий диапазон систем, совместной работы с другими приложениями на локальной и удаленных системах и взаимодействия с пользователями в стиле, облегчающем тем переход от системы к системе

Концепция открытых систем служит основным направлением при построении систем различного назначения — телекоммуникационных, вычислительных, реального времени. Особенно важно применять концепцию открытых систем и основанные на ней технические решения при создании информационных систем государственного, регионального муниципального и социального секторов народного хозяйства. Для разработки этого класса систем необходимо использовать технологии, обеспечивающие гарантированное соблюдение требований международных стандартов. Информационные технологии на базе открытых систем дают важные преимущества для всех категорий участников процесса информатизации: пользователей, разработчиков и производителей средств и систем информатизации. Основное достоинство концепции открытых систем — возможность сохранения и системного развития разработок, выполняемых отечественными предприятиями и организациями всех форм собственности за счет ранее сделанных инвестиций. Информационные технологии на базе концепции открытых систем реально обеспечат возможность самостоятельного развития отечественной вычислительной техники и информатики в условиях формирования рыночных отношений, интеграции России в мировое информационное сообщество и сохранения имеющегося интеллектуального, научно-технического и производственного потенциалов.

[В оглавление](#_top)

# Показатели качества ПС. Стандарты, регламентирующие показатели качества ПС.

На процессы разработки и оценки качества ПС оказывают влияние следующие обобщенные показатели ПС [4]:

1. область применения и назначение ПС;

2. тип решаемых функциональных задач;

3. объем и сложность ПС;

4. необходимый состав и требуемые значения характеристик качества

ПС и величина допустимого ущерба из-за недостаточного их качества;

5. степень связи решаемых задач с реальным масштабом времени или

допустимой длительностью ожидания результатов решения задачи;

6. прогнозируемые значения длительности эксплуатации и перспектива

создания множества версий ПС;

7. предполагаемый тираж производства и применения ПС;

8. степень необходимой документированности ПС.

Стандарт ISO 9126:1991 - Оценка программного продукта. Характеристики качества и руководство по их применению - является основой формального ***регламентирования характеристик качества***ПС. Развитие этого международного стандарта проводится в направлении уточнения, детализации и расширения, описаний характеристик качества комплексов программ. Для замены редакции 1991 года завершается разработка и формализован проект стандарта, состоящего из четырех частей **ISO 9126:1-4**. Стандарт ISO 9126:1991 предполагается заменить, на две взаимосвязанные серии стандартов: ISO 9126:1-4 (проект) - Качество программных средств - и утвержденный стандарт ISO 14598:1-6:1998-2000 - Оценивание программного продукта. Проект нового стандарта ISO 9126 состоит из следующих частей под общим заголовком - ***Информационная технология - Качество программных средств***:

Часть 1: Модель качества.

Часть 2: Внешние метрики качества.

Часть 3: Внутренние метрики качества.

Часть 4: Метрики качества в использовании.

Модель характеристик качества ПС состоит из шести групп базовых показателей, каждая из которых детализирована несколькими нормативными субхарактеристиками:

|  |  |
| --- | --- |
| ***Функциональная пригодность*** детализируется:   * пригодностью для применения; * корректностью (правильностью, точностью); * способностью к взаимодействию; * защищенностью.   ***Надежность*** характеризуется:   * уровнем завершенности (отсутствия ошибок); * устойчивостью к дефектам; * восстанавливаемостью; * доступностью-готовностью.   ***Эффективность*** рекомендуется отражать:   * временной эффективностью; * используемостью ресурсов. | ***Применимость*** (практичность) предлагается описывать:   * понятностью; * простотой использования; * изучаемостью; * привлекательностью.   ***Сопровождаемость*** представляется:   * удобством для анализа; * изменяемостью; * стабильностью; * тестируемостью.   ***Переносимость*** (мобильность) предлагается отражать:   * адаптируемостью; * простотой установки - инсталляции; * сосуществованием - соответствием; * замещаемостью. |

[В оглавление](#_top)

# Выбор и измерение показателей качества ПС.

**Выбор показателей качества**

Исходными данными и высшим приоритетом при выборе показателей качества в большинстве случаев являются назначение, функции и функциональная пригодность соответствующего программного средства. Принципиальные и технические возможности и точность измерения значений атрибутов характеристик качества всегда ограничены в соответствии с их содержанием. Это определяет рациональные диапазоны значений каждого атрибута, которые могут быть выбраны на основе здравого смысла, а также путем анализа прецедентов в спецификациях требований реальных проектов.

Процессы выбора и установления метрик и шкал для описания характеристик качества программных средств можно разделить на два этапа:

* выбор и обоснование набора исходных данных, отражающих общие особенности и этапы жизненного цикла проекта программного средства и его потребителей, каждый из которых влияет на определенные характеристики качества комплекса программ;
* выбор, установление и утверждение конкретных метрик и шкал измерения характеристик и атрибутов качества проекта для их последующей оценки и сопоставления с требованиями спецификаций в процессе квалификационных испытаний или сертификации на определенных этапах жизненного цикла программного средства.

Методологии и стандартизации оценки характеристик качества готовых программных средств и их компонентов (программного продукта) на различных этапах жизненного цикла посвящен международный стандарт ISO 14598, состоящий из шести частей. Рекомендуется следующая общая схема процессов оценки характеристик качества программ:

* установка исходных требований для оценки -- определение целей испытаний, идентификация типа метрик программного средства, выделение адекватных показателей и требуемых значений атрибутов качества;
* селекция метрик качества, установление рейтингов и уровней приоритета метрик субхарактеристик и атрибутов, выделение критериев для проведения экспертиз и измерений;
* планирование и проектирование процессов оценки характеристик и атрибутов качества в жизненном цикле программного средства;
* выполнение измерений для оценки, сравнение результатов с критериями и требованиями, обобщение и оценка результатов.
* Для каждой характеристики качества рекомендуется формировать меры и шкалу измерений с выделением требуемых, допустимых и неудовлетворительных значений. Реализация процессов оценки должна коррелировать с этапами жизненного цикла конкретного проекта программного средства в соответствии с применяемой, адаптированной версией стандарта ISO 12207.

**Функциональная пригодность** - наиболее неопределенная и объективно трудно оцениваемая субхарактеристика программного средства. Области применения, номенклатура и функции комплексов программ охватывают столь разнообразные сферы деятельности человека, что невозможно выделить и унифицировать небольшое число атрибутов для оценки и сравнения этой субхарактеристики в различных комплексах программ.

**Оценка корректности программных средств** состоит в формальном определении степени соответствия комплекса реализованных программ исходным требованиям контракта, технического задания и спецификаций на программное средство и его компоненты. Путем верификации должно быть определено соответствие исходным требованиям всей совокупности к компонентов комплекса программ, вплоть до модулей и текстов программ и описаний данных.

**Оценка способности к взаимодействию** состоит в определении качества совместной работы компонентов программных средств и баз данных с другими приклад­ными системами и компонентами на различных вычислительных платформах, а также взаимодействия с пользователями в стиле, удобном для перехода от одной вычислительной системы к другой с подобными функциями.

**Оценка защищенности программных средств**включает определение полноты использования доступных методов и средств защиты программного средства от потенциальных угроз и достигнутой при этом безопасности функционирования информационной системы. Наиболее широко и детально методологические и системные задачи оценки комплексной защиты информационных систем изложены в трех частях стандарта ISO 15408:1999-1--3 "Методы и средства обеспечения безопасности. Критерии оценки безопасности информационных технологий".

**Оценка надежности** - измерение количественных метрик атрибутов субхарактеристик в использовании: завершенности, устойчивости к дефектам, восстанавливаемости и доступности/готовности.

**Потребность в ресурсах памяти и производительности** компьютера в процессе решения за­дач значительно изменяется в зависимости от состава и объема ис­ходных данных. Для корректного определения предельной пропускной способности информа­ционной системы с данным программным средством нужно измерить экстремальные и средние значения длительностей исполнения функциональных групп программ и маршру­ты, на которых они достигаются. Если предварительно в процессе проектирования производи­тельность компьютера не оценивалась, то скорее всего понадобится большая доработка или даже заме­на компьютера на более быстродействующий.

**Оценка практичности** программных средств проводится экспертами и включает определение понятности, простоты использования, изучаемости и привлекательности программного средства. В основном это качественная (и субъективная) оценка в баллах, однако некоторые атрибуты можно оценить количественно по трудоемкости и длительности выполнения операций при использовании программного средства, а также по объему документации, необходимой для их изучения.

**Сопровождаемость** можно оценивать полнотой и достоверностью до­кументации о состояниях программного средства и его компонентов, всех предполагае­мых и выполненных изменениях, позволяющей установить текущее состояние версий программ в любой момент времени и историю их развития. Она должна опре­делять стратегию, стандарты, процедуры, распределение ресурсов и планы создания, изменения и применения документов на программы и данные.

**Оценка мобильности -**качественное определение экспертами адаптируемости, простоты установки, совместимости и замещаемости программ, выражаемое в баллах. Количественно эту характеристику программного средства и совокупность ее атрибутов можно (и целесообразно) оценить в экономических показателях: стоимости, трудоемкости и длительности реализации процедур переноса на иные платформы определенной совокупности программ и данных.

[В оглавление](#_top)

# Методы определения численных значений показателей качества ПС.

Частично присутствует в [предыдущем](#_Выбор_и_измерение) вопросе

[В оглавление](#_top)

# Применения метрик в управлении качеством ПС.

Статья из википедии. Следует воспользоваться старой доброй логикой, впрочем как и везде.

Ме́трика програ́ммного обеспе́чения ([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) software metric) — мера, позволяющая получить численное значение некоторого свойства [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) или его [спецификаций](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B5%D1%86%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F_%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).

Набор используемых метрик включает:

* порядок роста (имеется в виду анализ алгоритмов в терминах асимптотического анализа и O-нотации)
* количество строк кода
* цикломатическая сложность ( количество линейно независимых маршрутов через программный код)
* анализ функциональных точек
* количество ошибок на 1000 строк кода
* степень покрытия кода тестированием
* покрытие требований
* количество классов и интерфейсов
* связность

Метрики много критикуются, а вообще их часто используют для назначения з.п. прогерам-индусам (много строк => выше зп)

[В оглавление](#_top)

# Понятие сложности ПС. Основные компоненты сложности ПС.

Сложность вызывается четырьмя основными причинами:

* сложностью реальной предметной области, из которой исходит заказ на разработку;
* трудностью управления процессом разработки;
* необходимостью обеспечить достаточную гибкость программы;
* неудовлетворительными способами описания поведения больших дискретных систем.

**Сложность реального мира.** Проблемы, которые мы пытаемся решить с помощью программного обеспечения, часто неизбежно содержат сложные элементы, а к соответствующим программам предъявляется множество различных, порой взаимоисключающих требований. У пользователей и разработчиков разные взгляды на сущность проблемы, и они делают различные выводы о возможных путях ее решения. Знакомство с первыми версиями системы позволяет пользователям лучше понять и отчетливей сформулировать то, что им действительно нужно. В то же время процесс разработки повышает квалификацию разработчиков в предметной области и позволяет им задавать более осмысленные вопросы, которые проясняют темные места в проектируемой системе.

**Трудности управления процессом разработки.** Основная задача разработчиков состоит в создании иллюзии простоты, в защите пользователей от сложности описываемого предмета или процесса. Сегодня обычными стали программные системы, размер которых исчисляется десятками тысяч или даже миллионами строк на языках высокого уровня. Ни один человек никогда не сможет полностью понять такую систему. Поэтому такой объем работ потребует привлечения команды разработчиков. Чем больше разработчиков, тем сложнее связи между ними и тем сложнее координация, особенно если участники работ географически удалены друг от друга.

**Гибкость программного обеспечения**. Программирование обладает предельной гибкостью, и разработчик может сам обеспечить себя всеми необходимыми элементами, относящимися к любому уровню абстракции. Такая гибкость чрезвычайно соблазнительна. Она заставляет разработчика создавать своими силами все базовые строительные блоки будущей конструкции, из которых составляются элементы более высоких уровней абстракции. В отличие от строительной индустрии, где существуют единые стандарты на многие конструктивные элементы и качество материалов, в программной индустрии таких стандартов почти нет. Кроме того, часто приходится выполнять настройку программы под индивидуальные требования конкретного пользователя и системное окружение. Поэтому программные разработки остаются очень трудоемким делом.

**Проблема описания поведения больших дискретных систем**. Аналоговые системы, такие, как движение брошенного мяча, напротив, являются непрерывными. Небольшие изменения входных параметров всегда вызовут небольшие изменения выходных. С другой стороны, дискретные системы по самой своей природе имеют конечное число возможных состояний. Мы стараемся проектировать системы, разделяя их на части так, чтобы одна часть минимально воздействовало на другую. Каждое событие, внешнее по отношению к программной системе, может перевести ее в новое состояние, и, более того, переход из одного состояния в другое не всегда детерминирован. Всеобъемлющее тестирование таких программ провести невозможно. При неблагоприятных условиях небольшое внешнее событие может привести к критической ошибке системы.

[В оглавление](#_top)

# Показатели вычислительной сложности ПС.

Вычислительная сложность непосредственно связана с ресурсами вычислительной системы, необходимыми для получения совокупности законченных результатов. В ней выделяются три составляющие:

1. **Временная сложность** отражает необходимую длительность исполнения комплекса программ или время обработки на ЭВМ совокупности исходных данных до получения требуемых результатов. Временную сложность при обобщающих оценках измеряют количеством типовых операций, необходимых для получения результата. Такой подход позволяет исследовать характеристики сложности ПС независимо от быстродействия конкретной ЭВМ. В системах реального времени временная сложность непосредственно измеряется временем отклика (ожидания выходных результатов) на некоторые исходные данные.
2. **Программная сложность** характеризуется объемом памяти ЭВМ, необходимым для размещения ПС. В первом приближении программная сложность измеряется количеством команд, из которых она состоит. Однако различия количества используемых байт в памяти ЭВМ на одну команду затрудняют подсчет сложности программ по количеству команд, поэтому часто используют оценки по объему памяти, занимаемой ПС. Правда, такая оценка затрудняет сравнение программной сложности ПС, реализованных на ЭВМ, с различными структурами команд.
3. **Информационная сложность** определяется либо суммарным объемом баз данных, которые обрабатываются ПС, либо количеством типов и структурами данных, либо объемом оперативной и внешней памяти, используемой для накопления и хранения информации при выполнении программы.

Следует сделать следующее дополнение, которое касается временной сложности. Так как вычислительная сложность в наибольшей степени определяется алгоритмом решения задачи, а не реализующей его программой, то следует говорить о вычислительной сложности алгоритма. Поэтому по вычислительной сложности алгоритмы делятся на “хорошие” и “плохие”. К “хорошим” относятся алгоритмы, для которых скорость роста времени вычисления от размерности исходных данных пропорциональна nc, где c=const, c>1, а к “плохим” относятся алгоритмы, для которых скорость роста времени вычислений пропорциональна cn или еще больше.

[В оглавление](#_top)

# Измерение и оценка сложности ПС.

Для оценки сложности ПС используют метрики сложности.

Метрики сложности программ принято разделять на три основные группы:

* метрики размера программ;
* метрики сложности потока управления программ;
* метрики сложности потока данных программ.

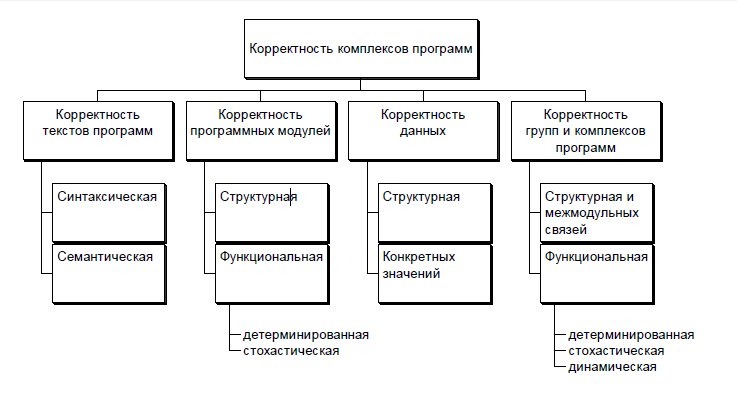
*Метрики первой группы* базируются на определении количественных характеристик, связанных с размером программы, и отличаются относительной простотой. К наиболее известным метрикам данной группы относятся число операторов программы, количество строк исходного текста, набор метрик Холстеда. Метрики этой группы ориентированы на анализ исходного текста программ. Поэтому они могут использоваться для оценки сложности промежуточных продуктов разработки.

*Метрики второй группы* базируются на анализе управляющего графа программы. Представителем данной группы является метрика Маккейба. Управляющий граф программы, который используют метрики данной группы, может быть построен на основе алгоритмов модулей. Поэтому метрики второй группы могут применяться для оценки сложности промежуточных продуктов разработки.

*Метрики третьей группы* базируются на оценке использования, конфигурации и размещения данных в программе. В первую очередь это касается глобальных переменных. К данной группе относятся метрики Чепина, Базили, Овьедо.

[В оглавление](#_top)

# Основные понятия и виды корректности программ.



Детерминированная корректность программ определяется по частности отклонения конкретных вычисляемых результатов от эталонных значений, заданных в техническом задании или в иных; исходных документах.

Стохастическая корректность характеризуется величиной статистического отклонения распределений и их параметров (средних значений, среднеквадратических отклонений) от заданных эталонов. При этом не оценивается каждый результат тестирования, а они обобщаются и оцениваются интегрально по некоторой достаточно представительной выборке.

Корректность в реальном времени (динамическая) определяется величиной максимального или усредненного отклонения траектории выходных параметров от заданной эталонной траектории обработки тестовых исходных данных. Результаты тестирования оцениваются интегрально на некотором временном интервале или по наибольшему отклонению от эталонной траектории.

[В оглавление](#_top)

# Типы эталонов, методы измерений и проверки корректности программ.



[В оглавление](#_top)Понятие ошибки в программе. Источники ошибок. Классификационная схема программных ошибок.

**Понятие ошибки в ПО**

Программное обеспечение содержит ошибку, если:

-поведение программного продукта не соответствует спецификациям, установленным при его разработке.

-программное обеспечение не соответствует официальной документации или не выполняет того, что пользователю разумно от него ожидать.

Учитывая разнообразие источников ошибок, при составлении плана тестирования классифицируют ошибки на два типа: 1 – синтаксические; 2 – семантические (смысловые).

Синтаксические ошибки – это ошибки в записи конструкций языка программирования (чисел, переменных, функций, выражений, операторов, меток, подпрограмм).

Семантические ошибки – это ошибки, связанные с неправильным содержанием действий и использованием недопустимых значений величин.

Источники ошибок:

Внутренние:

Ошибки проектирования

Ошибки алгоритмизации

Ошибки программирования

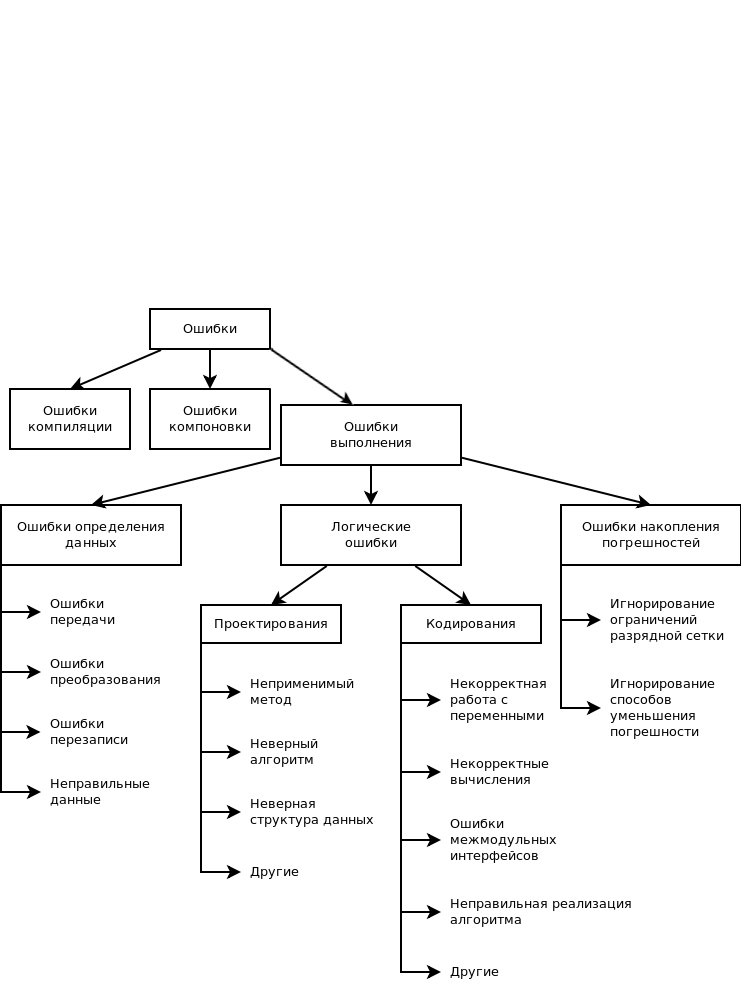
Внешние:

Ошибки пользователя

Сбои и отказы аппаратуры

Изменения конфигурации системы

Схема:



[В оглавление](#_top)

# Методы и средства тестирования и отладки программ.

Про тестирование написано [тут](#_Принципы_и_методы) и далее, до вопроса про отладку.

[В оглавление](#_top)

# Определение надежности ПС. Показатели надежности ПС.

**Надежность** - уровень завер­шенности (отсутствия ошибок), устойчивость к ошибкам и перезапускаемость.

Показатели надежности:

**Вероятность безотказной работы** – это вероятность того, что в пределах заданной наработки отказ системы не возникает.

**Вероятность отказа** – вероятность того, что в пределах заданной наработки отказ системы возникает.

**Интенсивность отказов системы** – это условная плотность вероятности возникновения отказа ПС в определенный момент времени при условии, что до этого времени отказ не возник.

**Средняя наработка до отказа** – математическое ожидание времени работы ПС до очередного отказа.

**Среднее время восстановления** – математическое ожидание времени восстановления.

**Коэффициент готовности** – вероятность того, что ПС ожидается в работоспособном состоянии в произвольный момент времени его использования по назначению.

[В оглавление](#_top)

# Факторы, определяющие надежность ПС.

Опыт создания и применения сложных информационных систем выявил множество ситуаций, при которых сбои и отказы функционирования были обусловлены дефектами программ и приводили к большим ущербам. Быстрый рост сфер использования, сложности и ответственности функций, выполняемых комплексами программ в информационных системах, резко повысил требования к их надежности и безопасности применения. Для выполнения этих требований в жизненном цикле (ЖЦ) программных средств (ПС) необходимо выделение задач и работ по обеспечению надежности ПО. В каждом проекте должен разрабатываться комплекс методов и средств обеспечения заданной надежности функционирования ПС.  
Одним из эффективных путей повышения надежности ПС является стандартизация технологических процессов и объектов проектирования, разработки и сопровождения программы.  
В стандартах ЖЦ ПС обобщается опыт и результаты исследования множества специалистов и рекомендуется наиболее эффективные методы и процессы.  
Для обнаружения и устранения ошибок проектирования все этапы разработки и сопровождения ПС д.б. поддержаны методами и средствами системного автоматического тестирования и испытания. Нужно применять различные методы и виды тестирования, каждый из которых ориентирован на обнаружение определенных видов дефектов.  
В сложных комплексах программ невозможно гарантировать абсолютное отсутствие ошибок, предсказать время и место появления ошибки, а также ее вид невозможно. Следовательно, необходимо создавать дополнительные системы автоматической оперативной защиты от непредумышленных случайных искажений программ и данных.  
Неравномерность распределения и нестабильность появления ошибок при использовании … не позволяет рассчитать их надежность классическими методами теории надежности технических систем. Реальная надежность функционирования сложных программных комплексов часто определяется экспериментально. В некоторых областях, например авиационной, нельзя использовать реальные объекты для тестирования и определения надежности ПС. В этих случаях особое значение приобрели методы и средства моделирования внешней среды для автоматизированной генерации тестов.

[В оглавление](#_top)

# Статические и динамические модели надежности ПС.

***ДИНАМИЧЕСКИЕ МОДЕЛИ***

|  |
| --- |
| ***Модель Шика - Волвертона.*** Модификация модели Джелинского - Моранды для случая возникновения на рассматриваемом интервале более одной ошибки предложена Волвертоном и Ши­ком. При этом считается, что исправление ошибок производится лишь после истечения интервала времени, на котором они воз­никли. В основе модели Шика - Волвертона лежит предположе­ние, «согласно которому частота ошибок пропорциональна не только количеству ошибок в программах, но и времени тестиро­вания, т.е. вероятность обнаружения ошибок с течением времени возрастает. Частота ошибок (интенсивность обнаружения оши­бок) предполагается постоянной в течение интервала времени ti и пропорциональна числу ошибок, оставшихся в программе по истечении (i-1)-го интервала; но она пропорциональна также и суммарному времени, уже затраченному на тестирование (вклю­чая среднее время выполнения программы в текущем интервале).  В данной модели наблюдаемым событием является число оши­бок, обнаруживаемых в заданном временном интервале, а не вре­мя ожидания каждой ошибки, как это было для модели Джелинского - Моранды. В связи с этим модель относят к группе диск­ретных динамических моделей.  ***Модель Муса.*** Модель Муса относят к динамическим моделям непрерывного времени. Это значит, что в процессе тестирования фиксируется время выполнения программы (тестового прогона) до очередного отказа. Но считается, что не всякая ошибка ПС может вызвать отказ, поэтому допускается обнаружение более одной ошибки при выполнении программы до возникновения очередного отказа.  В модели Муса различают два вида времени:  1) суммарное время функционирования т, которое учитывает чистое время тестирования до контрольного момента, когда про­водится оценка надежности;  2) оперативное время t — время выполнения программы, планируемое от контрольного момента и далее при условии, что дальнейшего устранения ошибок не будет (время безотказной работы в процессе эксплуатации).  Для суммарного времени функционирования т предполагается:  интенсивность отказов пропорциональна числу неустраненных ошибок;  скорость изменения числа устраненных ошибок, измеряемая относительно суммарного времени функционирования, пропорциональна интенсивности отказов.  Один из основных показателей надежности, который рассчи­тывается по модели Муса, — средняя наработка на отказ. Этот показатель определяется как математическое ожидание времен­ного интервала между последовательными отказами и связан с надежностью.  ***Модель переходных вероятностей***. Эта модель основана на марковском процессе, протекающем в дискретной системе с не­прерывным временем.  Процесс, протекающий в системе, называется марковским (или процессом без последствий), если для каждого момента времени вероятность любого состояния системы в будущем зависит толь­ко от состояния системы в настоящее время (to) и не зависит от того, каким образом система пришла в это состояние. Процесс тестирования ПС рассматривается как марковский процесс.  ***СТАТИЧЕСКИЕ МОДЕЛИ***  [Статические модели](http://info-tehnologii.ru/kac_sr/Mod_nad/index.html)прин­ципиально отличаются от динамических прежде всего тем, что в них не учитывается время появления ошибок в процессе тестиро­вания и не используется никаких предположений о поведении функции риска X(t). Эти модели строятся на твердом статисти­ческом фундаменте.  ***Модель Миллса.***Использование этой модели предполагает необходимость перед началом тестирования искусственно вно­сить в программу («засорять») некоторое количество известных ошибок. [Ошибки](http://info-tehnologii.ru/kac_sr/nad_sis/Obn_oshib/index.html)вносятся случайным образом и фиксируются в протоколе искусственных ошибок. Специалист, проводящий тес­тирование, не знает ни количества, ни характера внесенных оши­бок до момента оценки показателей надежности по модели Мил­лса. Предполагается, что все ошибки (как естественные, так и ис­кусственно внесенные) имеют равную вероятность быть найденными в процессе тестирования.  Тестируя программу в течение некоторого времени, собира­ют статистику об ошибках. В момент оценки надежности по протоколу искусственных ошибок все ошибки делятся на собствен­ные и искусственные.  ***Модель Липова.***Липов модифицировал модель Миллса, рас­смотрев вероятность обнаружения ошибки при использовании различного числа тестов. Если сделать то же предположение, что и в модели Миллса.  Модель Липова дополняет модель Миллса, давая возможность оценить вероятность обнаружения определенного количества ошибок к моменту оценки.  ***Простая интуитивная модель.***Использование этой модели предполагает проведение тестирования двумя группами програм­мистов (или двумя программистами в зависимости от величины программы) независимо друг от друга, использующими незави­симые тестовые наборы. В процессе тестирования каждая из групп фиксирует все найденные ею ошибки. При оценке числа остав­шихся в программе ошибок результаты тестирования обеих групп собираются и сравниваются.  Получается, что первая группа обнаружила ошибок, вто­рая — , а — это ошибки, обнаруженные обеими группами.  ***Модель Коркорэна****.*Модель Коркорэна относится к статичес­ким моделям надежности ПС, так как в ней не используются па­раметры времени тестирования и учитывается только результат *N*испытаний, в которых выявлено *N*ошибок i-го типа. Модель использует изменяющиеся вероятности отказов для различных типов ошибок. |

[В оглавление](#_top)

# Методы обеспечения технологической безопасности ПС и данных.

**В**со­временных автоматизированных технологиях создания и раз­вития сложных ПС с позиции обеспечения их необходимой и за­данной надежности можно выделить методы и средства, позво­ляющие:

• ***создавать программные модули и функциональные компоненты***высокого, гарантированного качества;

• ***предотвращать дефекты проектирования***за счет эффективных технологий и средств автоматизации обеспечения всего жиз­ненного цикла комплексов программ и баз данных;

• ***обнаруживать и устранять различные дефекты и ошибки***про­ектирования, разработки и сопровождения программ путем систематического тестирования на всех этапах жизненного цик­ла ПС;

• ***удостоверять достигнутое качество и надежность функциони­рования***ПС в процессе их испытаний и сертификации перед передачей в регулярную эксплуатацию;

• ***оперативно выявлять последствия дефектов программ и данных***и восстанавливать нормальное, надежное функционирование  
комплексов программ.

Комплексное, скоординированное применение этих методов и средств в процессе создания, развития и применения ПС позво­ляет исключать некоторые виды угроз или значительно ослаб­лять их влияние. Тем самым уровень достигаемой надежности ПС становится предсказуемым и управляемым, непосредственно зависящим от ресурсов, выделяемых на его достижение, а главное от качества и эффективности технологии, используемой на всех этапах жизненного цикла ПС.

Все принципы и методы обеспечения надежности в соответ­ствии с их целью можно разбить на четыре группы: ***предупреж­дение ошибок, обнаружение ошибок, исправление ошибок***и ***обеспе­чение устойчивости к ошибкам.*К**первой группе относятся прин­ципы и методы, позволяющие минимизировать или вообще исключить ошибки. Методы второй группы сосредоточивают внимание на функциях самого программного обеспечения, помо­гающих выявлять ошибки. К третьей группе относятся функции программного обеспечения, предназначенные для исправления ошибок или их последствий. Устойчивость к ошибкам (четвертая группа) — это мера способности системы программного обеспе­чения продолжать функционирование при наличии ошибок.

[В оглавление](#_top)

# Эмпирические модели надежности ПС.

[Эмпирические модели](http://info-tehnologii.ru/kac_sr/Mod_nad/index.html)в основном базируются на анализе структурных особенностей программного средства (или програм­мы). Как указывалось ранее, эмпирические модели часто не дают конечных результатов показателей надежности, однако их использование на этапе проектирования ПС полезно для прогнозиро­вания требующихся ресурсов тестирования, уточнения плановых сроков завершения проекта и т.д.

Модель сложности. В литературе неоднократно подчеркива­ется тесная взаимосвязь между сложностью и надежностью ПС. Если придерживаться упрощенного понимания сложности ПС, то она может быть описана такими характеристиками, как раз­мер ПС (количество программных модулей), количество и слож­ность межмодульных интерфейсов.

Под программным модулем в данном случае следует понимать программную единицу, выполняющую определенную функцию (ввод, вывод, вычисление и т.д.) взаимосвязанную с другими модулями ПС. Сложность модуля ПС может быть описана, если рассматривать структуру программы.

В качестве структурных характеристик модуля ПС использу­ются:

1) отношение действительного числа дуг к максимально возможному числу дуг, получаемому искусственным соединением каждого узла с любым другим узлом дугой;

2) отношение числа узлов к числу дуг;

3) отношение числа петель к общему числу дуг.

Для сложных модулей и для больших многомодульных про­грамм составляется имитационная модель, программа которой «засоряется» ошибками и тестируется по случайным входам. [Оцен­ка надежности](http://info-tehnologii.ru/kac_sr/nad_sis/Poc_nad/Nad_fun/index.html)осуществляется по модели Миллса.

При проведении тестирования известна структура програм­мы, имитирующей действия основной, но не известен конкрет­ный путь, который будет выполняться при вводе определенного тестового входа. Кроме того, выбор очередного тестового набо­ра из множества тест-входов случаен, т.е. в процессе тестирова­ния не обосновывается выбор очередного тестового входа. Эти условия вполне соответствуют реальным условиям тестирования больших программ.

Полученные данные анализируются, проводится расчет по­казателей надежности по модели Миллса (или любой другой из описанных выше), и считается, что реальное ПС, выполняющее аналогичные функции, с подобными характеристиками и в ре­альных условиях должно вести себя аналогичным или похожим образом.

[Преимущества оценки показателей надежности](http://info-tehnologii.ru/kac_sr/nad_sis/Poc_nad/index.html)по имитаци­онной модели, создаваемой на основе анализа структуры буду­щего реального ПС, заключаются в следующем:

• модель позволяет на этапе проектирования ПС принимать оптимальные проектные решения, опираясь на характеристики ошибок, оцениваемые с помощью имитационной модели;

• модель позволяет прогнозировать требуемые ресурсы тестирования;

• модель дает возможность определить меру сложности программ и предсказать возможное число ошибок и т.д.

К недостаткам можно отнести высокую стоимость метода, так как он требует дополнительных затрат на составление имитаци­онной модели, и приблизительный характер получаемых показа­телей.

[В оглавление](#_top)

# Методы сбора и обработки данных о разработках ПС.

Выбор методов для создания средств автоматизации процесса прогнозирования ТЭП по критерию точности весьма затруднителен. Это вызвано прежде всего

различиями условий сбора и обработки статистических данных о разработках ПС. В связи с этим в качестве основных критериев отбора методов были приняты:

•    возможность использования методов как на начальных, так и на более поздних этапах разработки;

•    апробирование методов в отечественной и зарубежной практике;

•    возможность приведения результатов их применения к сопоставимому виду.

В наибольшей степени этим критериям отвечают известные по публикациям методы:

•    базовая модель Б. Боэма (модель 1);

•    промежуточная модель Б. Боэма. объединенная с моделью Л. Путнама (модель 2);

•    модель обобщенного аналога В. Липаева (модель 3).

[В оглавление](#_top)

# Задачи и проблемы сертификации ПС.

При анализе процессов сертификационных испытаний ПС следует выделить ряд базовых компонент методологии сертификации, подлежащих последующему рассмотрению:

цели сертификации формальные, технологические, правовые, экономические;

проблемы, которые необходимо решать для обеспечения высокой эффективности и достоверности результатов сертификационных испытаний ИТ, ПС и БД;

исходные данные и документы, необходимые для проведения сертификации стандарты и нормативные документы, их структура и содержание;

характеристики и классификация программ и БД как объектов испытаний и сертификации, их показатели качества, позволяющие выделять однородные группы ПС и БД при проведении сертификации;

ресурсы обеспечения испытаний финансовые, кадры специалистов, аппаратурная оснащенность, нормативно-технические и программно-инструментальные средства.

При анализе сертификации ИТ, ПС и БД целесообразно выделить следующие проблемы:

Научно-методические, состоящие в создании эффективных по затратам ресурсов методов сертификационных испытаний ИТ, ПС и БД, которые гарантируют достоверное определение заданных показателей их качества и соответствие документации;

технологические, заключающиеся в обеспечении реализации методов испытаний ИТ средствами автоматизации, тестирования и организации регламентированных проверок качества объектов и документации на разных этапах их создания и при непосредственных сертификационных испытаниях;

проблемы стандартизации и нормативной документации

организационные, состоящие в создании международных, государственных и ведомственных органов, ответственных за сертификацию ИТ

экономические, которые сводятся к выявлению, оценке и применению экономически эффективных методов использования ресурсов испытаний ПС.

правовые, сосредоточивающие в себе прежде всего создание юридических механизмов процессов сертификации и использования их результатов, создание нормативов, правил взаимодействия и распределения экономической и юридической ответственности между разработчиками, производителями, сертифицирующими организациями и поставщиками ПС.

[В оглавление](#_top)

# Виды сертификационных испытаний и программ.

**Сертификационные испытания** – элемент системы мероприятий, направленных на подтверждение соответствия фактических характеристик изделия требованиям нормативно технической документации (НТД). **Сертификационные испытания**, как правило, проводят независимые от производителя испытательные центры аккредитованные в различных системах сертификации. По результатам испытаний выдается сертификат или знак соответствия изделия требованиям НТД. Сертификация предполагает взаимное признание результатов испытаний поставщиком и потребителем.

**Программу и методы испытаний** устанавливают в сертификационной документации и указывают в положении по сертификации данного изделия с учетом особенностей его изготовления.

Известны следующие виды испытаний ПС, проводимых с целью аттестации ПС:  
  
- испытания компонент ПС;  
  
- системные испытания;  
  
- приемо-сдаточные испытания;  
  
- полевые испытания;  
  
- промышленные испытания.

Дополнительный материал есть в [этом](#_Испытания_и_сертификация) вопросе

[В оглавление](#_top)

# Методы, технология, средства обеспечения сертификации программных средств.

Материал есть в [этом](#_Испытания_и_сертификация) вопросе

[В оглавление](#_top)

# Особенности современных программных средств и баз данных как объектов разработки.

Повтор [этого](#_Специфические_особенности_ПС.) вопроса

[В оглавление](#_top)

# Цели и принципы документирования программных средств. Стандарты документирования программных средств.

Цели и общие слова [тут](#_Документирование_ПС._Состав)

Стандарты документирования:

ГОСТ Р ИСО 9127-94 «Документация пользователя и информация на упаковке для потребительских программных пакетов». Довольно кратко (весь документ – около 20 страниц) указаны основные требования к составу и содержанию документации пользователя.

ГОСТ Р ИСО/МЭК 15910-2002 «Процесс создания документации пользователя программного средства» — стандарт больше отвечает не на вопрос «Что» должно быть в документе, а «Как» должен создаваться документ. Дополнительно есть подробное описание стиля документа с примером – довольно удобная штука для создания шаблона: один раз запариваешься (и забиваешь в шаблон всё: от выравнивания до формата подписей рисунков), а потом клепаешь документы все одного вида, а не с заголовками разного шрифта.

ГОСТ-ы серии 19.хх – серия Единой системы программной документации (в среднем, документы созданы в 78-м году), в них такие же лаконичные, как в ГОСТ 9127, требования ко многим видам документов.

ГОСТ 34.602-89 «Техническое задание на создание автоматизированной системы» — стандарт на ТЗ.

ГОСТ 34.201-89 Виды, комплектность и обозначения документов при создании автоматизированных систем. Это базовый документ, в котором приводится полный перечень документации ГОСТ 34, рекомендации по кодированию документов, к каким стадиям проекта относятся документы (стадии описываются в ГОСТ 34.601-90), а также как их можно объединить между собой.

[В оглавление](#_top)

# Технологическая и эксплуатационная документация на ПС.

Документация [тут](#_Документирование_ПС._Состав). Технологическая, эта та, которой пользуются для создания, а эксплуатационной для использования (т.е. пользовательская)

[В оглавление](#_top)

# Организация документирования программных средств. Управление документированием этапов жизненного цикла ПС.

 В общем случае должны быть ***выделены руководители и коллектив специалистов***, которые будут планировать, описывать, утверждать, выпускать, распрост­ранять и сопровождать комплекты документов. Официальная, описанная и утвержденная стратегия документирования должна ***устанавливать дисциплину***, необходимую для эффективного создания высо­кокачественных документов на продукты и процессы в жизненном цик­ле ПС.

На начальном этапе проектирования следует выделять и формировать целесообразный ***комплект шаблонов документов,*** обеспечивающих регла­ментирование всех этапов, процессов и документов при создании опре­деленных проблемно-ориентированных проектов ПС.

Процессы документирования программ и данных входят в весь жизненный цикл сложных систем и ПС. Поэтому организация и реализация ***работ по созданию документов должны распределяться между специа­листами***, ведущими непосредственное и преимущественное создание проектов комплексов программ и специалистами осуществляющими, в основном, разработ­ку, контроль и издание документов.

Базой эффективного управления проектом ПС и его документированием должен быть ***План***, в котором задачи исполнителей частных работ согласованы с выделяемыми для них ресурсами, а также между собой по результатам и срокам их достижения. План проекта должен отражать рациональное сочетание целей, стратегий действий, конкретных процедур, доступных ресурсов и других ком­понентов, необходимых для достижения основной цели с заданным качеством.

Необходимо следить за ходом проекта и документирования на всем протяжении жизненного цикла и сравнивать запланиро­ванные и фактические результаты работ и документы. ***Контроль яв­ляется органической функцией управления*** и должен иметь ряд средств ре­гулирования поведения отдельных специалистов и коллектива разработчиков документов в целом.

***При планировании и разработке комплекс документации должен проверяться и аттестовываться*** на полноту в условиях ограниченных ресурсов, на корректность, адекватность и непротиворечивость отдельных документов.

Особое внимание в последнее время уделяется ***совершенс­твованию и детализации документов, обеспечивающих высокое ка­чество создаваемых ПС***, а также возможности их эффективного итерационно­го развития длительное время в многочисленных версиях. Соответственно долж­ны изменяться документы, отражающие состояние процессов и компо­нентов проектов. Для этого организация процессов документирования должна обеспечивать гибкое и точное изменение документов – ***сопровожде­ние и конфигурационное управление версиями и редакциями каждого документа***.

Для хранения, тиражирования и распространения документов, сложных ПС высокого качества, следует выделять группу специалис­тов, ответственных за ***контроль, обеспечение и гарантированное сохранение документации***. Кроме того, должна быть организована ***служба нормативного конт­роля***, ответственная за соблюдение стандартов, нормативных и ру­ководящих документов при подготовке документации всеми специа­листами, участвующими в крупном проекте. Эта служба обязана обеспечить унификацию и высокое качество содержания, структуры и оформле­ния шаблонов документов.

Для обеспечения достоверных данных об объектах и процессах управления документами ПС, необходима автоматизированная ***база данных***-***информаци­онная система обеспечения и хранения документов проекта***.

**В плане управления документированием каждого этапа жизнен­ного цикла ПС целесообразно фиксировать и документально оформлять**:

* исходные данные (шаблоны), требующиеся для успешного выполнения дан­ного этапа документирования проекта или компонента ПС;
* контролируемые и документируемые данные о состоянии объекта и процесса разработки, регистрируемые после завершения этапа;
* содержание процедур контроля состояния проекта и документов в процессе выполнения работ этапа;
* критерии оценки результатов выполненных работ и качества отчетных документов при завершении этапа;
* состав и содержание отчетных документов (шаблонов), представляемых для оценки состояния проекта, результатов завершенного этапа и работ и для использования на следующем этапе или при завершении проек­та ПС.

[В оглавление](#_top)

# Документация управления качеством ПС.

Документация [здесь](#_Документирование_ПС._Состав), и вся документация управления разработки используется для управления качеством. Типа организация рабочего процесса => меньше ошибок => выше качество

[В оглавление](#_top)

# Структура и содержание документов по этапам жизненного цикла ПС.

1. **Виды программных документов и их содержание.**

[*Спецификация*](http://podgoretsky.com/ftp/Docs/Formats/uspd/uspd3.htm#00) - содержит состав программы и документации на нее. Выполняется на стадии рабочего проекта.

*Ведомость держателей подлинников* - содержит перечень предприятий, на которых хранят подлинники программных документов. Выполняется на стадии рабочего проекта.

[*Текст программы*](http://podgoretsky.com/ftp/Docs/Formats/uspd/uspd3.htm#12)  - содержит запись программы с необходимыми комментариями. Выполняется на стадии рабочего проекта.

[*Описание программы*](http://podgoretsky.com/ftp/Docs/Formats/uspd/uspd3.htm#13)  - содержит сведения о логической структуре и функционировании программы. Выполняется на стадии рабочего проекта.

[*Программа и методика испытаний*](http://podgoretsky.com/ftp/Docs/Formats/uspd/uspd3.htm#51) - содержит требования, подлежащие проверке при испытаниях программы, а также порядок и методы их контроля. Выполняется на стадии рабочего проекта.

[*Техническое задание*](http://podgoretsky.com/ftp/Docs/Formats/uspd/uspd3.htm#01) - содержит назначение и область применения программы, технические, технико-экономические и специальные требования, предъявляемые к программе, необходимые стадии и сроки разработки, виды испытаний. Выполняется на стадии технического задания.

[*Пояснительная записка*](http://podgoretsky.com/ftp/Docs/Formats/uspd/uspd3.htm#81)  - содержит схему алгоритма, общее описание алгоритма и /или функционирования программы, а также обоснование принятых технических и технико-экономических решений. Выполняется на стадии эскизного и технического проектов.

*Эксплуатационные документы* - содержит сведения для обеспечения функционирования и эксплуатации программы. Выполняются на стадии рабочего проекта.

1. **Виды эксплуатационных документов и их содержание**

*Ведомость эксплуатационных документов*  - содержит перечень эксплуатационных документов на программу.

*Формуляр* - содержит основные характеристики программы, комплектность и сведения об эксплуатации программы.

[*Описание применения*](http://podgoretsky.com/ftp/Docs/Formats/uspd/uspd3.htm#31) - содержит сведения о назначении программы, области применения, применяемых методах, классе решаемых задач, ограничениях для применения, минимальной конфигурации технических средств.

[*Руководство системного программиста*](http://podgoretsky.com/ftp/Docs/Formats/uspd/uspd3.htm#32) - содержит сведения для проверки, обеспечения функционирования и настройки программы на условия конкретного применения.

[*Руководство программиста*](http://podgoretsky.com/ftp/Docs/Formats/uspd/uspd3.htm#33) - содержит сведения для эксплуатации программы.

[*Руководство оператора*](http://podgoretsky.com/ftp/Docs/Formats/uspd/uspd3.htm#34) - содержит сведения для обеспечения процедуры общения оператора с вычислительной системой в процессе выполнения программы.

[*Описание языка*](http://podgoretsky.com/ftp/Docs/Formats/uspd/uspd3.htm#35)  - содержит описание синтаксиса и семантики языка.

[*Руководство по техническому обслуживанию*](http://podgoretsky.com/ftp/Docs/Formats/uspd/uspd3.htm#46)  - содержит сведения для применения тестовых и диагностических программ при обслуживании технических средств.

1. **Стадии разработки программ, программной документации, а также этапы и содержание работ:**

|  |  |  |
| --- | --- | --- |
| Стадии разработки | Этапы работ | Содержание работ |
| Техническое задание | Обоснование необходимости разработки программы | Постановка задачи. Сбор исходных материалов. Выбор и обоснование критериев эффективности и качества разрабатываемой программы.  Обоснование необходимости проведения научно-исследовательских работ. |
| Научно-исследовательские работы | Определение структуры входных и выходных данных. Предварительный выбор методов решения задач. Обоснование целесообразности применения ранее разработанных программ. Определение требований к техническим средствам. Обоснование принципиальной возможности решения поставленной задачи. |
| Разработка и утверждение технического задания | Определение требований к программе. Разработка технико-экономического обоснования разработки программы. Определение стадий, этапов и сроков разработки программы и документации на нее. Выбор языков программирования. Определение необходимости проведения научно-исследовательских работ на последующих стадиях. Согласование и утверждение технического задания. |
| Эскизный проект | Разработка эскизного проекта | Предварительная разработка структуры входных и выходных данных. Уточнение методов решения задачи. Разработка общего описания алгоритма решения задачи. Разработка технико-экономического обоснования. |
| Утверждение эскизного проекта | Разработка пояснительной записки. Согласование и утверждение эскизного проекта |
| Технический проект | Разработка технического проекта | Уточнение структуры входных и выходных данных. Разработка алгоритма решения задачи. Определение формы представления входных и выходных данных. Определение семантики и синтаксиса языка. Разработка структуры программы. Окончательное определение конфигурации технических средств. |
| Утверждение технического проекта | Разработка плана мероприятий по разработке и внедрению программ. Разработка пояснительной записки. Согласование и утверждение технического проекта. |
| Рабочий проект | Разработка программы | Программирование и отладка программы |
| Разработка программной документации | Разработка программных документов в соответствии с требованиями ГОСТ 19.101-77. |
| Испытания программы | Разработка, согласование и утверждение программы и методики испытаний. Проведение предварительных государственных, межведомственных, приемо-сдаточных и других видов испытаний. Корректировка программы и программной документации по результатам испытаний. |
| Внедрение | Подготовка и передача программы | Подготовка и передача программы и программной документации для сопровождения и (или) изготовления. Оформление и утверждение акта о передаче программы на сопровождение и (или) изготовление. Передача программы в фонд алгоритмов и программ. |

[В оглавление](#_top)

# Состав пользовательской документации на ПС.

Рассмотрено [здесь](#_Документирование_ПС._Состав)

[В оглавление](#_top)

# Техническое задание на проектирование ПС. Эскизный (технический), рабочий проект ПС.

**Техническое задание (ТЗ)** — исходный документ разработки информационных систем.   
ТЗ содержит основные технические требования, предъявляемые к программному обеспечению, его составу, средствам реализации, надежности и отказоустойчивости,  а также условиям его эксплуатации.

Проектирование программного обеспечения — процесс создания проекта программного обеспечения (ПО), а также дисциплина, изучающая методы проектирования. Проектирование ПО является частным случаем Проектирования продуктов и процессов.

Целью проектирования является определение внутренних свойств системы и детализации её внешних (видимых) свойств на основе выданных заказчиком требований к ПО (исходные условия задачи). Эти требования подвергаются анализу.  
Первоначально программа рассматривается как чёрный ящик. Ход процесса проектирования и его результаты зависят не только от состава требований, но и выбранной модели процесса, опыта проектировщика.

Модель предметной области накладывает ограничения на бизнес-логику и структуры данных.

В зависимости от класса создаваемого ПО, процесс проектирования может обеспечиваться как «ручным» проектированием, так и различными средствами его автоматизации. В процессе проектирования ПО для выражения его характеристик используются различные нотации — блок-схемы, ER-диаграммы, UML-диаграммы, DFD-диаграммы, а также макеты.

Проектированию обычно подлежат:

* Архитектура ПО;
* Устройство компонентов ПО;
* Пользовательские интерфейсы.

[В оглавление](#_top)

# Документация тестирования компонентов и комплексов программ. Документация испытаний комплексов программ.

ПРИЕМО-СДАТОЧНЫЕ ИСПЫТАНИЯ

|  |
| --- |
| *Организация завершающих испытаний комплексов программ. Испытания главного конструктора,*которые зачастую совмеща­ются с завершением комплексной отладки, должны оформляться документально и являются основанием для предъявления ПС за­казчику на завершающиеся совместные испытания. Любые испы­тания ограничены допустимым объемом проверок и длительнос­тью работы комиссии, поэтому не могут гарантировать абсолют­ную проверку изделия. Для повышения достоверности опреде­ления и улучшения характеристик ПС после испытаний главного конструктора программы целесообразно передавать некоторым пользователям на *опытную эксплуатацию в типовых условиях.*Это позволяет более глубоко оценить эксплуатационные характерис­тики созданного комплекса и устранить некоторые дефекты и ошибки. Опытная эксплуатация проводится разработчиками с уча­стием испытателей и некоторых пользователей, назначаемых за­казчиком. Результаты и [показатели надежности](http://info-tehnologii.ru/kac_sr/nad_sis/Poc_nad/index.html)опытной эксплуа­тации после испытаний главного конструктора могут учитывать­ся при проведении совместных испытаний для их сокращения.  Программа испытаний является планом проведения серии эк­спериментов и разрабатывается с позиции минимизации объема тестирования в процессе проведения испытаний для проверки выполнения требований технического задания и соответствия предъявленной документации. Программа испытаний, методики их проведения и оценки результатов, разработанные совместно заказчиком и разработчиком, должны быть согласованы и ут­верждены. Они должны содержать уточнения и детализацию тре­бований технического задания для данного ПС, а также гаранти­ровать корректную проверку всех заданных характеристик, в том числе надежности. *Программа испытаний должна содержать сле­дующие четко сформулированные разделы:*  • объект испытаний, его назначение и перечень основных документов, определивших его разработку;  • цель испытаний с указанием всех требований технического задания, подлежащих проверке, и ограничений на проведение ис­пытаний;  • собственно программу испытаний, содержащую проверку комплектности спроектированного ПС в соответствии с техническим заданием, и план тестирования для проверки по всем разделам технического задания и дополнительным требованиям, формали­зованным отдельными решениями разработчиков и заказчика;  • методики испытаний, однозначно определяющие все понятия проверяемых характеристик, условия и сценарии тестирования, средства, используемые для испытаний;  • методики обработки и оценки результатов тестирования по каждому разделу программы испытаний. |

[В оглавление](#_top)

# Документация сопровождения и конфигурационного управления версиями программ.

Система управления версиями — программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.

Такие системы наиболее широко используются при разработке программного обеспечения для хранения исходных кодов разрабатываемой программы. Однако они могут с успехом применяться и в других областях, в которых ведётся работа с большим количеством непрерывно изменяющихся электронных документов. В частности, системы управления версиями применяются в САПР, обычно в составе систем управления данными об изделии (PDM). Управление версиями используется в инструментах конфигурационного управления.

Традиционные системы управления версиями используют централизованную модель, когда имеется единое хранилище документов, управляемое специальным сервером, который и выполняет бо́льшую часть функций по управлению версиями. Пользователь, работающий с документами, должен сначала получить нужную ему версию документа из хранилища; обычно создаётся локальная копия документа, так называемая «рабочая копия». Может быть получена последняя версия или любая из предыдущих, которая может быть выбрана по номеру версии или дате создания, иногда и по другим признакам. После того, как в документ внесены нужные изменения, новая версия помещается в хранилище. В отличие от простого сохранения файла, предыдущая версия не стирается, а тоже остаётся в хранилище и может быть оттуда получена в любое время. Сервер может использовать т. н. дельта-компрессию — такой способ хранения документов, при котором сохраняются только изменения между последовательными версиями, что позволяет уменьшить объём хранимых данных. Поскольку обычно наиболее востребованной является последняя версия файла, система может при сохранении новой версии сохранять её целиком, заменяя в хранилище последнюю ранее сохранённую версию на разницу между этой и последней версией. Некоторые системы (например, ClearCase) поддерживают сохранение версий обоих видов: большинство версий сохраняется в виде дельт, но периодически (по специальной команде администратора) выполняется сохранение версий всех файлов в полном виде; такой подход обеспечивает максимально полное восстановление истории в случае повреждения репозитория.

[В оглавление](#_top)

# Общая характеристика инструментальных средств документирования. Пакеты программ для формирования отчетов.

Документация является органической, составной частью программного продукта для ЭВМ и требуются значительные ресурсы для ее создания и применения. Тексты и объектный код программ для ЭВМ могут стать программным продуктом только в совокупности с комплексом документов, полностью соответствующих их содержанию и достаточных для его освоения, применения и изменения.

К числу ПС, которые могут быть использованы практически в любой сфере деятельности, относятся текстовые редакторы, табличные процессоры, интегрированные ПС, автопереводчики, системы оптического распознавания и т.п. Указанные средства необходимы для подготовки и редактирования научной, технической документации и прочих текстовых документов, ведения делопроизводства, расчета смет и калькуляций, ведомостей форм бухгалтерской отчетности и т. п.

Пакеты программ:

|  |  |
| --- | --- |
| Платные | Бесплатные |
| Microsoft Windows, Mac OC | Ubuntu, Linux |
| MC Office | OpenOffice, Libre Office |
| WinRar, WinZip | 7Zip |
| Adobe photoshop | Paint.net, Gimp |

[В оглавление](#_top)

# Пакеты конфигурационного управления.

Есть в [этом](#_Сопровождение_и_конфигурационное) вопросе

[В оглавление](#_top)