МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Пятигорский институт (филиал) СКФУ

Колледж Пятигорского института (филиал) СКФУ

УТВЕРЖДАЮ

Директор Пятигорского
института
(филиал) СКФУ
Т.А. Шебзухова

ПМ.02 ОСУЩЕСТВЛЕНИЕ ИНТЕГРАЦИИ ПРОГРАММНЫХ МОДУЛЕЙ

МДК 02.01 ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ЛАБОРАТОРНЫХ РАБОТ

Специальности СПО

09.02.07 Информационные системы и программирование

Пятигорск, 2025г.

Методические указания для лабораторных работ по дисциплине МДК 02.01 Технология разработки программного обеспечения составлены в соответствии с требованиями ФГОС СПО. Предназначены для студентов, обучающихся по специальности 09.02.07 Информационные системы и программирование.

ЛАБОРАТОРНАЯ РАБОТА № 1

Тема: Предпосылки становления дисциплины.
Профессионализм в программировании. Программные продукты (изделия): определения и классификация

Цель: ознакомиться с технологией разработки технического задания.

Исходные данные (задание):

1. Ознакомиться с ГОСТ 19.201-78 «Техническое задание. Требования к содержанию и оформлению».
2. Ознакомиться с примерами оформления технического задания.
3. Проанализировать предложенные примеры технических заданий на соответствие ГОСТ 19.201-78.
4. Результаты анализа оформить в виде таблицы.

ГОСТ 19.201-78 Единая система программной документации (ЕСПД). Техническое задание. Требования к содержанию и оформлению (с Изменением N 1)

ГОСТ 19.201-78

Группа Т55

МЕЖГОСУДАРСТВЕННЫЙ СТАНДАРТ

Единая система программной документации

ТЕХНИЧЕСКОЕ ЗАДАНИЕ. ТРЕБОВАНИЯ К СОДЕРЖАНИЮ И ОФОРМЛЕНИЮ

Unified system for program documentation. Technical specification for development. Requirements for contents and form of pressentation

МКС 35.080

Дата введения 1980-01-01

Постановлением Государственного комитета СССР по стандартам от 18 декабря 1978 г. N 3351 дата введения установлена 01.01.80

ИЗДАНИЕ (январь 2010 г.) с Изменением N 1, утвержденным в июне 1981 г. (ИУС 9-81).

Настоящий стандарт устанавливает порядок построения и оформления технического задания на разработку программы или программного изделия для вычислительных машин, комплексов и систем независимо от их назначения и области применения.

Стандарт полностью соответствует СТ СЭВ 1627-79*.

_____

* Доступ к международным и зарубежным документам, упомянутым здесь, можно получить, перейдя по ссылке на сайт http://shop.cntd.ru. - Примечание изготовителя базы данных.
(Измененная редакция, Изм. N 1).

1. ОБЩИЕ ПОЛОЖЕНИЯ

1. ОБЩИЕ ПОЛОЖЕНИЯ

1.1. Техническое задание оформляют в соответствии с ГОСТ 19.106-78 на листах формата 11 и 12 по ГОСТ 2.301-68, как правило, без заполнения полей листа. Номера листов (страниц) проставляют в верхней части листа над текстом.

1.2. Лист утверждения и титульный лист оформляют в соответствии с ГОСТ 19.104-78. Информационную часть (аннотацию и содержание), лист регистрации изменений допускается в документ не включать.

1.3. Для внесения изменений или дополнений в техническое задание на последующих стадиях разработки программы или программного изделия выпускают дополнение к нему. Согласование и утверждение дополнения к техническому заданию проводят в том же порядке, который установлен для технического задания.

1.4. Техническое задание должно содержать следующие разделы:

введение;

основания для разработки;

назначение разработки;

требования к программе или программному изделию;

требования к программной документации;

технико-экономические показатели;

стадии и этапы разработки;

порядок контроля и приемки;

в техническое задание допускается включать приложения.

В зависимости от особенностей программы или программного изделия допускается уточнять содержание разделов, вводить новые разделы или объединять отдельные из них.

(Измененная редакция, Изм. N 1).

2. СОДЕРЖАНИЕ РАЗДЕЛОВ

2.1. В разделе "Введение" указывают наименование, краткую характеристику области применения программы или программного изделия и объекта, в котором используют программу или программное изделие.

2.2. В разделе "Основание для разработки" должны быть указаны:
документ (документы), на основании которых ведется разработка;
организация, утвердившая этот документ, и дата его утверждения;
наименование и (или) условное обозначение темы разработки.

2.1, 2.2 (Измененная редакция, Изм. N 1).

2.3. В разделе "Назначение разработки" должно быть указано функциональное и эксплуатационное назначение программы или программного изделия.

2.4. Раздел "Требования к программе или программному изделию" должен содержать следующие подразделы:
требования к функциональным характеристикам;
требования к надежности;
условия эксплуатации;
требования к составу и параметрам технических средств;
требования к информационной и программной совместимости;
требования к маркировке и упаковке;
требования к транспортированию и хранению;
специальные требования.
(Измененная редакция, Изм. N 1).

2.4.1. В подразделе "Требования к функциональным характеристикам" должны быть указаны требования к составу выполняемых функций, организации входных и выходных данных, временным характеристикам и т.п.

2.4.2. В подразделе "Требования к надежности" должны быть указаны требования к обеспечению надежного функционирования (обеспечение устойчивого функционирования, контроль входной и выходной информации, время восстановления после отказа и т.п.).

2.4.3. В подразделе "Условия эксплуатации" должны быть указаны условия эксплуатации (температура окружающего воздуха, относительная влажность и т.п. для выбранных типов носителей данных), при которых должны обеспечиваться заданные характеристики, а также вид обслуживания, необходимое количество и квалификация персонала.

2.4.4. В подразделе "Требования к составу и параметрам технических средств" указывают необходимый состав технических средств с указанием их основных технических

характеристик.

2.4.5 В подразделе "Требования к информационной и программной совместимости" должны быть указаны требования к информационным структурам на входе и выходе и методам решения, исходным кодам, языкам программирования и программным средствам, используемым программой.

При необходимости должна обеспечиваться защита информации и программ.
(Измененная редакция, Изм. N 1).

2.4.6. В подразделе "Требования к маркировке и упаковке" в общем случае указывают требования к маркировке программного изделия, варианты и способы упаковки.

2.4.7. В подразделе "Требования к транспортированию и хранению" должны быть указаны для программного изделия условия транспортирования, места хранения, условия хранения, условия складирования, сроки хранения в различных условиях.

2.5а. В разделе "Требования к программной документации" должны быть указаны предварительный состав программной документации и, при необходимости, специальные требования к ней.
(Введен дополнительно, Изм. N 1).

2.5. В разделе "Технико-экономические показатели" должны быть указаны: ориентировочная экономическая эффективность, предполагаемая годовая потребность, экономические преимущества разработки по сравнению с лучшими отечественными и зарубежными образцами или аналогами.

2.6. В разделе "Стадии и этапы разработки" устанавливают необходимые стадии разработки, этапы и содержание работ (перечень программных документов, которые должны быть разработаны, согласованы и утверждены), а также, как правило, сроки разработки и определяют исполнителей.

2.7. В разделе "Порядок контроля и приемки" должны быть указаны виды испытаний и общие требования к приемке работы.

2.8. В приложениях к техническому заданию, при необходимости, приводят:
перечень научно-исследовательских и других работ, обосновывающих разработку;
схемы алгоритмов, таблицы, описания, обоснования, расчеты и другие документы, которые могут быть использованы при разработке;
другие источники разработки.

I. Пример оформления технического задания

1. Общие сведения

1.1. Полное наименование АИС: Информационная система по автоматизации учёта товаров в магазине «Меридиан».
1.2. Краткое наименование системы: АИС – «Меридиан».
1.3. Основания для проведения работ

Разработка ведется на основании договора №1 от 09.11.12 между заказчиком (Сафиуллин И. А. директор «Меридиан») и разработчиком (Ганиева Г. Д.)

Заказчик: ООО «Меридиан»

Адрес фактический: 453587, РБ, Белорецкий р-н, д. Азикеево, ул. Нугуш д.49. Телефон: +7 (347)9248392

Разработчик: ООО «АСУ»

Адрес фактический: РБ, г. Белорецк, ул. К. Маркса 51.

Телефон: +7 (347) 9233343

Факс: +7 (347)92 45347

1.4. Сроки разработки: (на период 2013 - 2014 гг.).

1.5. Источники и порядок финансирования

Финансирование работ по созданию АИС будет осуществляться заказчиком. Результаты работ по созданию ИС или ее частей оформляются разработчиком в письменном виде и предоставляются в заранее оговоренные сроки.

2. Назначение и цели создания АИС

2.1. Назначение системы

АСУ предназначена для автоматизации управления деятельностью фирмы, а именно:

- Учет материальных запасов
- Учет основных средств и нематериальных активов
- Формирование отчетов
- Учет движения товаров на складе

2.2. Цели создания системы

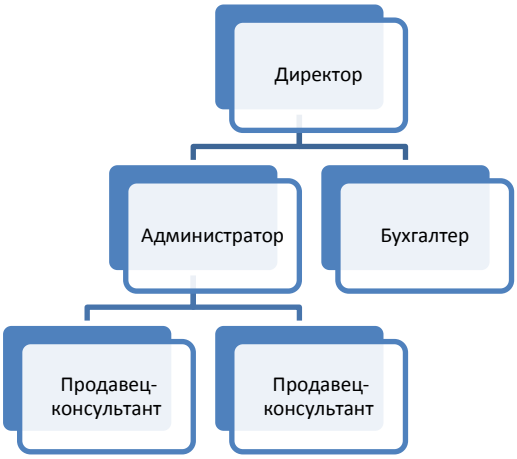
Целями создания ИС является повышение качества работы «Меридиан», она также должна облегчить и ускорить сбор, обработку и хранение информации.

3. Характеристика объекта автоматизации

Ассортимент ООО «Меридиан» состоит из широкого выбора товаров разных категорий, однако большая часть приходится на продовольственные товары.

Магазин – пункт розничной торговли, занимает отдельное помещение и имеет торговый зал для покупателей. В функции склада входят создание необходимого ассортимента для выполнения заказов покупателей. Накопление и сохранение позволяет осуществлять непрерывное возобновление и поставку на базе созданных товарных запасов. Склад используется как промежуточное звено в цепочке производитель-покупатель.

В силу проведенного обследования выяснилось необходимо установить на два уже имеющихся компьютера систему. На одном из компьютеров (главный компьютер) при помощи ИС и будет вестись учет движения товаров и денежных средств, полученных от их реализации. Специалист, которого назначат для работы в магазине, будет заносить данные о приходе товара на склад, о его расходе и перемещениях со склада в торговый зал магазина. В конце каждой недели он должен будет сформировать отчет об остатках товаров на складе и проверить соответствие данных в компьютере с наличием товаров на складе и в торговом зале. Он будет полностью нести ответственность за соответствие данных в компьютере с иными данными на складе. Второй компьютер будет установлен в торговом зале, продавцы, могут сами списывать товар с базы, если этот товар находится в торговом зале. А если товара нет в их распоряжении, то они только будут принимать заказ от покупателя, а списывать товар с наличия в базе будет специалист (т.к. он полностью отражает все перемещения товаров в торговый зал).



Организационная диаграмма (рис.1)

В соответствии с организационной структурой деятельность магазина условно подразделена на основную и вспомогательную деятельность. Но поскольку пользователями информационной системы будет персонал, выполняющий основную деятельность, то остановимся подробнее на функциях, выполняемых именно этими людьми.

Директор осуществляет торгово-закупочную деятельность и реализацию товаров в розничной торговой сети. Прием на работу сотрудников, увольнение, кадровые перемещения штатное расписание, а также формы, размеры и оплата труда оформляются приказами директора.

Бухгалтер организует бухгалтерский отчёт и отчётность, принимает решения по другим вопросам, связанным с текущей деятельностью общества.

Администратор следит за состоянием выкладки товаров и рекламного оформления торгового зала, за соблюдением правил торговли, принимает решения по претензиям покупателей.

Продавцы выполняют всю работу с покупателями. Консультируют их по товару, подводят итоги получаемой прибыли в конце рабочего дня.

Коллектив магазина «Меридиан» обязан соблюдать правила торговли, обеспечивать сохранность продукции, обеспечивать соблюдение правил и норм охраны труда и техники безопасности, производственной санитарии и противопожарной безопасности.

В магазине выполняются следующие основные операции: прием, размещение, хранение товара; ее подготовка на продажу; оформление приходных и расходных документов.

Рассмотрим факторы, которые смогут повлиять на результат работы исследуемого нами отдела. В качестве основной методологии проектирования на данном этапе будем использовать **методологию ARIS, диаграмму причин и факторов Исикавы** (рис. 2)

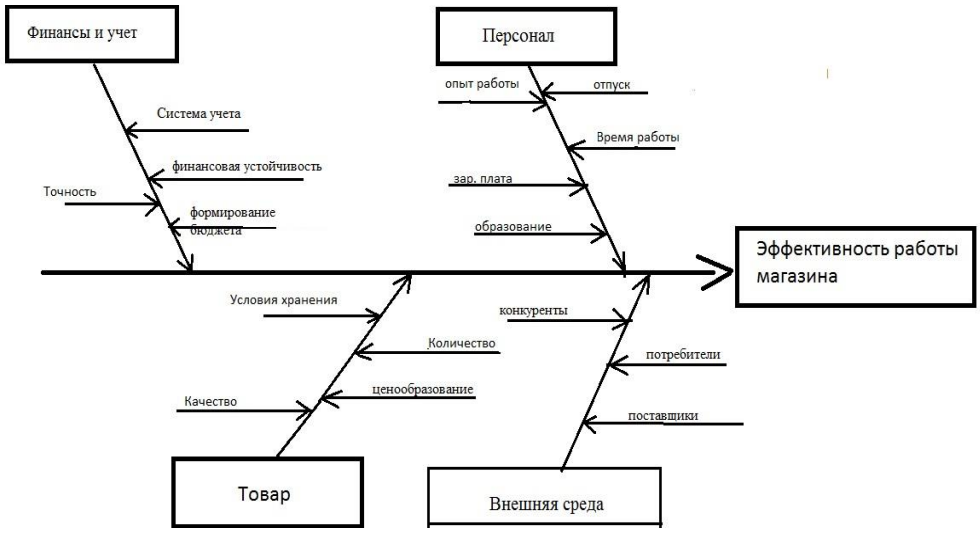


Диаграмма Исикавы (рис.2)

Проведем анализ построенной диаграммы.

В начале анализа выделим **«показатель качества»** - «Эффективность работы магазина», который является основным.

Далее обозначим факторы**,** оказывающие на него непосредственное влияние.

К ним можно отнести: рабочий персонал; финансы и учет; товар; внешняя среда. Рассмотрим каждый фактор подробнее.

Рабочий персонал – это непосредственные работники магазина. На данный фактор влияет следующее: квалификация, образование, опыт сотрудника, время работы, отпуск, заработная плата.

Учет – это то, каким образом на складе организован учет размещенного товара. На данный фактор влияет точность учета; форма учета; способ учета.

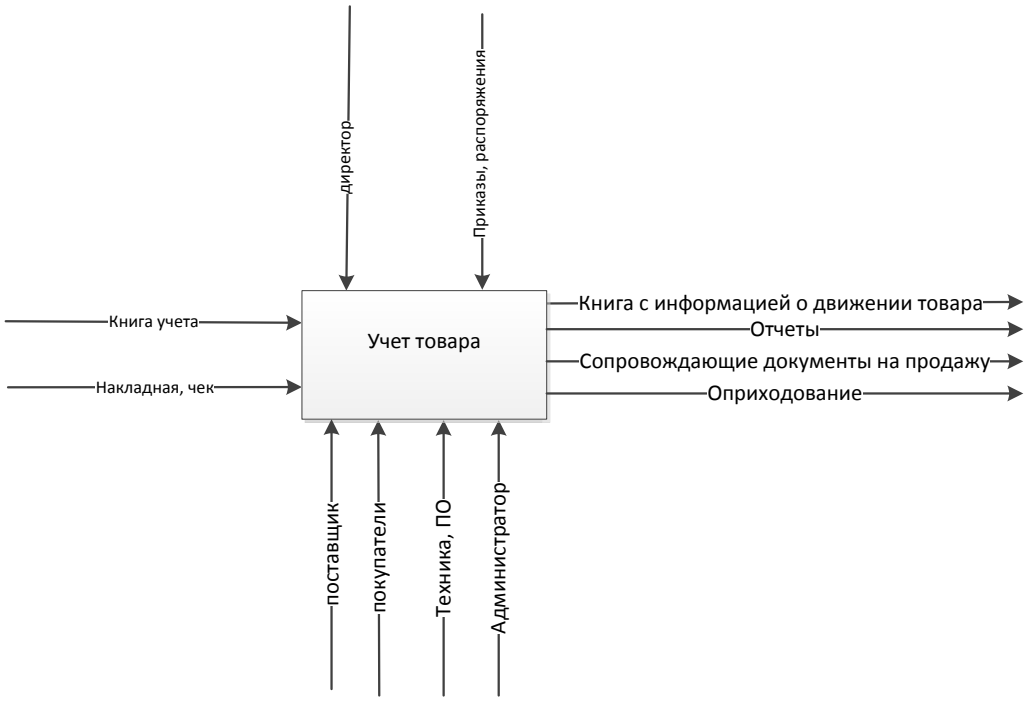
Товар – это непосредственный предмет хранения. На данный фактор влияет следующее: качество (изначальное состояние, условия и время хранения); количество (объемы поставок, количество).

Внешняя среда – это все условия и факторы, возникающие в окружающей среде, независимо от деятельности конкретной фирмы, но оказывающие или могущие оказать воздействие на её функционирование и поэтому требующие принятия управленческих решений. Это поставщики, потребители, конкуренты, законодательство и тд.

При дальнейшем анализе рассматриваемой предметной области необходимо учитывать влияние каждого фактора, как главного, так и второстепенного, поскольку их негативное воздействие может привести к неудовлетворительному качеству хранения товара и, как следствие, к плохой репутации магазина.

Функциональное моделирование позволяет описать алгоритм процесса, то есть последовательность функций. Во-первых, магазин представляется в виде совокупности бизнесов процессов – это последовательность шагов, которое обеспечивает его функциональную деятельность. Во-вторых, взаимодействие функций реализуется через виды стрелок (вход, выход – рис.3).

Для функциональной модели выбирается точка зрения среднего звена, так как сотрудники могут реализовывать и видеть весь процесс.



Контекстная диаграмма (рис. 3)

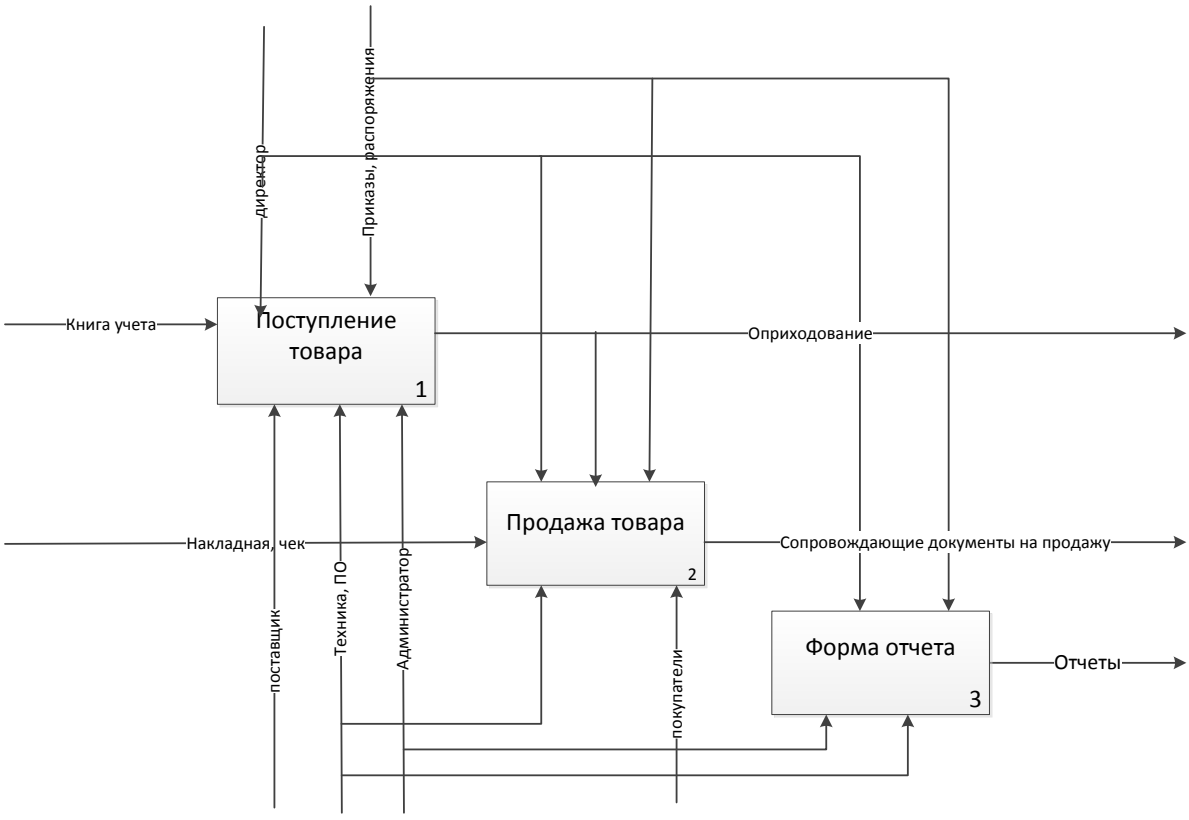
Отчет по диаграмме IDEF0

Название модели: Учет товара и денежных потоков в магазине ООО «Мередиан».

Описание: учет товара и денежных потоков.

Точка зрения: директор ООО «Мередиан»

Цель: уменьшить трудовые, временные и финансовые ресурсы по обороту товара в магазине.



Декомпозиция контекстной диаграммы (рис. 4)

Товар привозится в магазин и принимается по накладной. Далее в учетной системе магазина создается документ прихода.

Продажи товара в учетную систему заносятся вручную или формируются автоматически, как "документ продажи товара" при продаже его через кассовый узел.

4 Требования к ИС

Требования:

- Система должна быть простой и понятной пользователю.
- Внедрение ИС должно привести к положительному экономическому эффекту.

Основные функции, которые должна выполнять система:

- Ведение учета товаров в магазине
- Планирование затрат, связанных с приобретением и хранением товаров
- Планирование доходов, связанных с реализацией товаров

Требования к видам обеспечения ИС:

- Техническое обеспечение должно составлять комплекс технических средств соединенных в локальную сеть.
- Программное обеспечение должно включать:

1. 1С «Предприятие» и компоненты для работы ИС
2. Антивирусные программы
3. Офисные программы (MS Office или Open Office)

- Информационное обеспечение должно включать данные о товарах, поставщиках, ценах.
- Программное обеспечение представляет собой совокупность правовых норм регламентирующих правоотношения при создании и внедрении системы. Правовое обеспечение на этапе разработки должно включать нормативные акты, с правовым регулированием отношений в ходе этого процесса.
- Даная информационная система будет внедрена в течении 3,5–4 месяцев.
- Для работы с ИС необходимо обучить персонал.

2.5 Состав и содержание работ по созданию системы

Предпроектная стадия включает в себя:

- Определение требований заказчика;
- Разработка проекта АИС в соответствии с требованиями заказчика;
- Разработка технического задания в соответствии с ГОСТ 34.602–89.27.01.08–21.02.08;

Проектная стадия:

- Внедрение ИС;
- Сопровождение системы.

Исполнителями работ являются:

- Разработчик ИС;
- Специалист по созданию локальной сети;
- Специалист по установке, конфигурированию, сопровождению системы.

Ответственный за выполнение всех работ по всем этапам является разработчик ИС.

2.6 Порядок контроля и приема системы

Чтобы заказчик принял систему, необходимо провести пробную эксплуатацию, во время которой ведут журнал, где записываются все решения задач и все нарушения. По

результатам эксплуатации составляют протокол, в который вносят недостатки, определяют сроки устранения.

Во время приема директором должны быть представлены документы:

- Техническое задание на систему
- Технический и рабочий проекты на систему
- Протокол и журнал испытательной эксплуатации
- Штатное расписание подразделений заказчика, которые обслуживают систему
- Акты передачи всех частей информационной системы заказчику.

2.7 Требования к составу и содержанию работ по подготовке объекта автоматизации к вводу в эксплуатацию

Для подготовки объекта к вводу в эксплуатацию следует выполнить:

- Подготовка объекта к переходу на работу в новой ИС
- Опробование всех материалов технического и рабочего проектов и внесение изменений по результатам

Для внедрения ИС в эксплуатацию необходимо:

- Оформить акты о выполнении плана мероприятий по подготовке объекта к внедрению.
- Наличие документации по внедрению ИС.
- Наличие персонала, что обеспечивает подготовку внедрения и эксплуатацию.
- Наличие принятых в эксплуатацию технических средств ИС.

2.8 Требования к документации

- Документация оформляется согласно стандартам единой системе программной документации и ГОСТ.

Кроме того, оформляются и согласовываются разработчиком и заказчиком, различного рода договора на проведение работ, акты о выполнении этапов создания ИС, графики выполнения работ по этапам и документы, составленные после выполнения каждого этапа.

2.9 Источники разработки

- Техническое задание на создание системы
- Техническое задание на автоматизированные системы управления

также используется информация, полученная от управленческого и рабочего персонала, и на основании требований заказчика.

1. Техническое задание на информационный представительский сайт с каталогом продукции
2. Техническое задание на разработку сайта для кондитерского комбината
3. Техническое задание на создание туристического сайта
4. Техническое задание на создание представительского сайта (информационного ресурса) газеты
5. Техническое задание на интернет-магазин мебели
6. Техническое задание на интернет-магазин цветов
7. Техническое задание на разработку сайта торгового представительства с каталогом продукции и прайс-листом

ЛАБОРАТОРНАЯ РАБОТА № 2

Тема: Примеры программных проектов. Факты о программном обеспечении.

Цель: сформировать техническое задание по предложенной информационной системе.

Разработка технического задания

Исходные данные (задание):

1. Ознакомиться с ГОСТ 19.201-78 «Техническое задание. Требования к содержанию и оформлению».
2. Сформировать техническое задание по информационной системе.
3. Оформить техническое задание.

Ниже приведено 15 вариантов программных продуктов. По указанию преподавателя выберите свое индивидуальное задание. Разработайте техническое задание на создание программного продукта по всем требованиям.

1. Разработка программного комплекса «Автотранспорт».
2. Разработка программного комплекса «Деканат института».
3. Разработка программного комплекса «Обслуживание банкомата».
4. Разработка программного комплекса «Управление гостиницей».
5. Разработка программного комплекса «Выдача кредитов в банке».
6. Разработка программного комплекса «Строительная фирма».
7. Разработка программного комплекса «Управление библиотечным фондом».
8. Разработка программного комплекса «АРМ работника склада»
9. Разработка программного комплекса «АРМ администратора ателье по ремонту оргтехники»
10. Разработка программного комплекса «АРМ администратора автосалона».
11. Разработка программного комплекса «АРМ администратора ресторана».
12. Разработка программного комплекса «АРМ сотрудника ЖЭСа».
13. Разработка программного комплекса «АРМ администратора аэропорта».
14. Разработка программного комплекса «АРМ работника отдела кадров».
15. Разработка программного комплекса «АРМ администратора спорткомплекса».

ЛАБОРАТОРНАЯ РАБОТА № 3

Тема: Классический жизненный цикл. Реальный классический жизненный цикл.

Цель: Ознакомление со стандартами в области обеспечения жизненного цикла программных средства. Рассмотреть стадии создания ПС. Опишите особенности каскадной модели жизненного цикла ПС. Рассмотреть жизненный цикл АС. Перечислите этапы работ согласно ГОСТ 19.102-77 «Стадии разработки программ и программной документации».

Введение

В основе деятельности по созданию и использованию программных средств лежит понятие жизненного цикла. Жизненный цикл является моделью создания и использования программного обеспечения, отражающей его различные состояния, начиная с момента

возникновения необходимости в программном средстве и заканчивая моментом его полного выхода из употребления у пользователей.

Основными целями применения стандартов и нормативных документов в жизненном цикле ПС являются:

- снижение трудоемкости, длительности, стоимости и улучшение других технико-экономических показателей проектов ПС;
- повышение качества разрабатываемых и/или применяемых компонентов и ПС в целом при их приобретении, разработке, эксплуатации и сопровождении;
- обеспечение возможности расширять ПС по набору прикладных функций и масштабировать в зависимости от размерности решаемых задач;
- обеспечение переносимости прикладных программ и данных между разными аппаратно-программными платформами.

Применение стандартов позволяет ориентироваться на построение систем из крупных функциональных узлов, отвечающих требованиям стандартов, применять отработанные и проверенные проектные решения. Они определяют унифицированные интерфейсы и протоколы взаимодействия компонентов таким образом, что разработчику системы, как правило, не требуется вдаваться в детали внутреннего устройства этих компонентов.

В нашей стране жизненный цикл разработки ПС установлен стандартом ГОСТ 19.102-77 «Стадии разработки программ и программной документации» и содержит следующие этапы работ:

- техническое задание (ТЗ);
- эскизный проект (ЭЗ);
- технический проект (ТП);
- рабочий проект (РП);
- внедрение.

В таблице 3 приведены стадии разработки и этапы, их составляющие.

Таблица 3. Стадии и этапы разработки ПС

Стадии разработки

Этапы работ

Техническое задание

Обоснование необходимости разработки программы

Научно-исследовательские работы

Разработка и утверждение технического задания

Эскизный проект

Разработка эскизного проекта

Утверждение эскизного проекта

Технический проект

Разработка технического проекта

Утверждение технического проекта

Рабочий проект

Разработка программы

Разработка программной документации

Испытания программы

Внедрение

Подготовка и передача программы

Кроме рассмотренного выше жизненного цикла программ, существует жизненный цикл автоматизированных систем (АС) ГОСТ 34.601–90 «Информационная технология. Автоматизированные системы. Стадии создания». Настоящий стандарт распространяется

на автоматизированные системы, используемые в различных видах деятельности (исследование, проектирование, управление и т. п.), включая их сочетания, создаваемые в организациях, объединениях и на предприятиях. Стандарт устанавливает стадии и этапы создания АС, а также содержание работ на каждом этапе.

Процесс создания АС представляет собой совокупность упорядоченных во времени, взаимосвязанных, объединенных в стадии и этапы работ, выполнение которых необходимо и достаточно для создания АС, соответствующей заданным требованиям (табл. 4).

Допускается исключение стадии «Эскизный проект» и отдельных этапов работ на всех стадиях, объединение стадий «Технический проект» и «Рабочая документация» в одну стадию «Техно-рабочий проект».

Таблица 4. Стадии и этапы разработки АС

Наименование этапа

Содержание этапа

1.Формирование требований к АС

Обследование объекта и обоснование необходимости создания АС.

Формирование требований пользователя АС.

Оформление отчета о выполненной работе и заявки на разработку АС (тактико-технического задания)

2. Разработка концепции АС

Изучение объекта.

Проведение необходимых научно-исследовательских работ.

Разработка вариантов концепции АС и выбор варианта концепции АС, удовлетворяющего требованиям пользователя.

Оформление отчета о выполненной работе

3. Техническое задание

Разработка и утверждение технического задания на создание АС

4. Эскизный проект

Разработка предварительных проектных решений по системе в целом и ее частям.

Разработка документации на АС и ее части

5. Технический проект

Разработка проектных решений по системе и ее частям.

Разработка документации на АС и ее части.

Разработка и оформление документации на поставку изделий для комплектования АС и/или технических требований (технических заданий) на их разработку.

Разработка заданий на проектирование в смежных частях проекта объекта автоматизации

6. Рабочая документация

Разработка рабочей документации на систему и ее части.

Разработка или адаптация программ

В зависимости от специфики создаваемых АС и условий их создания допускается выполнение отдельных этапов работ до завершения предшествующих стадий, параллельное выполнение этапов работ, включение новых этапов работ.

Стандарт ISO 12207 (ГОСТ Р ИСО/МЭК 12207) «Информационная технология. Процессы жизненного цикла программных средств» наиболее полно на уровне международных стандартов отражает жизненный цикл, технологию разработки и обеспечения качества сложных программных средств. Жизненный цикл ПС представлен набором этапов, частных работ и операций в последовательности их выполнения и взаимосвязи, регламентирующих ведение разработки на всех стадиях от подготовки

технического задания до завершения испытаний ряда версий и окончания эксплуатации ПС. В жизненный цикл включаются описания исходной информации, способов выполнения операций и работ, устанавливаются требования к результатам и правилам их контроля, а также к содержанию технологических и эксплуатационных документов. Определяется организационная структура коллективов, распределение и планирование работ, а также контроль за реализацией жизненного цикла ПС.

Стандарт может использоваться как непосредственный директивный, руководящий или рекомендательный документ, а также как организационная база при создании средств автоматизации соответствующих технологических этапов или процессов. Для реализации положений стандарта должны быть выбраны инструментальные средства, совместно образующие взаимосвязанный комплекс технологической поддержки и автоматизации ЖЦ и не противоречащие предварительно скомпонованному набору нормативных документов. Имеющиеся в стандарте пробелы следует заполнять спецификациями или нормативными документами, регламентирующими применение выбранных или созданных инструментальных средств автоматизации разработки и документирования ПС.

Задание 1. Рассмотреть ГОСТ 19.102-77 «Стадии разработки программ и программной документации», выписать основные стадии разработки ПС, цели каждого этапа.

Задание 2. Рассмотреть ГОСТ 19.102-77 «Стадии разработки программ и программной документации». Перечислите этапы работ по созданию АС. Согласно ГОСТа, определить каскадный и спиральный метод создания ПС.

Задание 3. Выполнить сравнение каскадной и спиральной модели создания ПС.

ЛАБОРАТОРНАЯ РАБОТА № 4

Тема: Стратегии разработки ПО

Цель: Изучить стратегии разработки программного обеспечения, выполнить анализ и сравнение.

Сравнительный анализ стратегий разработки ПО

1. Ознакомиться с теоретической частью.
2. Выполнить практическое задание.
3. Ответить на контрольные вопросы.
4. Оформить отчет.

Теоретическая часть

Базовые стратегии разработки программных средств и систем

На начальном этапе развития вычислительной техники ПС разрабатывались по принципу «кодирование – устранение ошибок». Модель такого процесса разработки ПС иллюстрирует рисунок 1.

Очевидно, что недостатками данной модели являются:

- неструктурированность процесса разработки ПС;
- ориентация на индивидуальные знания и умения программиста;
- сложность управления и планирования проекта;
- большая длительность и стоимость разработки;
- низкое качество программных продуктов;
- высокий уровень рисков проекта.

Для устранения или сокращения вышеназванных недостатков созданы и широко используются три базовые стратегии разработки ПО: каскадная, инкрементная, эволюционная.

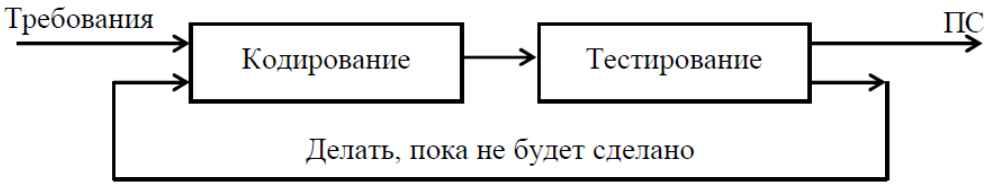


Рисунок 1 – Модель «Делать, пока не будет сделано»

Некоторые характеристики каскадной, инкрементной и эволюционной стратегий разработки ПС и предъявляемые к ним требования приведены в стандарте ГОСТ Р ИСО/МЭК ТО 15271–2002 – Информационная технология – Руководство по применению ГОСТ Р ИСО/МЭК 12207 (Процессы жизненного цикла программных средств).

Выбор той или иной стратегии определяется характеристиками: проекта, требований к продукту, команды разработчиков, команды пользователей. Три базовые стратегии могут быть реализованы с помощью различных моделей ЖЦ.

Каскадная стратегия разработки программных средств и систем

Каскадная стратегия представляет собой однократный проход этапов разработки. Данная стратегия основана на полном определении всех требований к разрабатываемому программному средству или системе в начале процесса разработки. Каждый этап разработки начинается после завершения предыдущего этапа. Возврат к уже выполненным этапам не предусматривается. Промежуточные продукты разработки в качестве версии программного средства (системы) не распространяются. Представителями моделей, реализующих каскадную стратегию, являются каскадная и V-образная модели.

Инкрементная стратегия разработки программных средств и систем

Инкрементная стратегия представляет собой многократный проход этапов разработки с запланированным улучшением результата.

Данная стратегия основана на полном определении всех требований к разрабатываемому программному средству (системе) в начале процесса разработки. Однако полный набор требований реализуется постепенно в соответствии с планом в последовательных циклах разработки.

Результат каждого цикла называется инкрементом.

Эволюционная стратегия разработки программных средств и систем

Эволюционная стратегия представляет собой многократный проход этапов разработки. Данная стратегия основана на частичном определении требований к разрабатываемому программному средству или системе в начале процесса разработки. Требования постепенно уточняются в последовательных циклах разработки. Результат каждого цикла разработки обычно представляет собой очередную поставляемую версию программного средства или системы.

Практическая часть

1. Запишите определение каскадной стратегии разработки ПО.
2. Выделите основные достоинства и недостатки каскадной стратегии.
3. Приведите область применения каскадной модели.
4. Запишите определение инкрементной стратегии разработки ПО.
5. Запишите основные достоинства и недостатки инкрементной стратегии.
6. Выделите область применения инкрементной модели.
7. Запишите определение эволюционной стратегии разработки ПО.
8. Выделите основные достоинства и недостатки эволюционной стратегии.
9. Приведите область применения эволюционной модели.

10. Выполните сравнение стратегий и запишите результаты в сводную таблицу.

Характеристика проекта	Стратегия		
	Каскадная	**Инкрементная**	**Спиральная**
Новизна разработки и обеспеченность ресурсами			
Масштаб проекта			
Срок выполнения проекта			
Заключение отдельных договоров на отдельные версии			
Определение основных требований в начале проекта			
Изменение требований по мере развития проекта			
Разработка итерациями			
Распространение промежуточного ПС			

11. Результаты выполнения практического задания запишите в отчет.

Контрольные вопросы

1. Дайте определение понятию «программная инженерия».
2. Дайте определение понятию «жизненный цикл».
3. Что такое макетирование? Изобразите схему данного процесса.
4. Выберите подходящий процесс разработки для перечисленных ниже программных приложений. Обоснуйте свой выбор.
1. Система решения квадратных уравнений.
2. Система определения оценки по результатам ответа на три экзаменационных вопроса.
3. Информационная система института.

Содержание отчета

1. Тема.
2. Цель.
3. Оборудование.
4. Результат выполнения практического задания.
5. Ответы на контрольные вопросы.
6. Вывод.

ЛАБОРАТОРНАЯ РАБОТА № 5

Тема: Изучение достоинства и недостатки спиральной модели разработки ПО.

Цель: Изучить достоинства и недостатки спиральной модели разработки ПО.

Изучение достоинств и недостатков спиральной модели разработки ПО

Спиральная модель жизненного цикла информационной системы (предложена в 1986 году Барри Боэмом) – процесс разработки программного обеспечения, сочетающий в себе как итеративность, так и этапность. Главное отличие этой модели заключается в

уделении особого внимания рискам, влияющим на организацию жизненного цикла. Боэм формулирует десять наиболее распространенных рисков:

1. дефицит специалистов;
2. нереалистичные сроки и бюджет;
3. реализация несоответствующей функциональности;
4. разработка неправильного пользовательского интерфейса;
5. перфекционизм, ненужная оптимизация;
6. непрекращающийся поток изменений;
7. нехватка информации о внешних компонентах;
8. недостатки в работах, выполняемых внешними ресурсами;
9. недостаточная производительность получаемой системы;
10. разрыв между квалификацией специалистов и требованиями проекта.

Большая часть этих рисков связана с постоянно изменяющимися условиями разработки и функционирования информационной системы, поэтому их нельзя избежать без изменения самой системы. 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. При таком подходе детали проекта последовательно углубляются и конкретизируются и в результате выбирается обоснованный вариант, который доводится до реализации.

Каждый виток разбивается на 4 сектора: 1. оценка и разрешение рисков; 2. определение целей; 3. разработка и тестирование; 4. планирование.

На рис. 1 показано графическое представление спиральной модели.

Неполное завершение работ на каждом этапе позволяет переходить на следующий этап, не дожидаясь полного завершения на текущем. При интерактивном способе разработки недостающую работу можно будет выполнить на следующей итерации. Главная задача спиральной модели – как можно быстрее показать пользователям системы работоспособный продукт, тем самым анализируя процесс уточнения и дополнения требований.

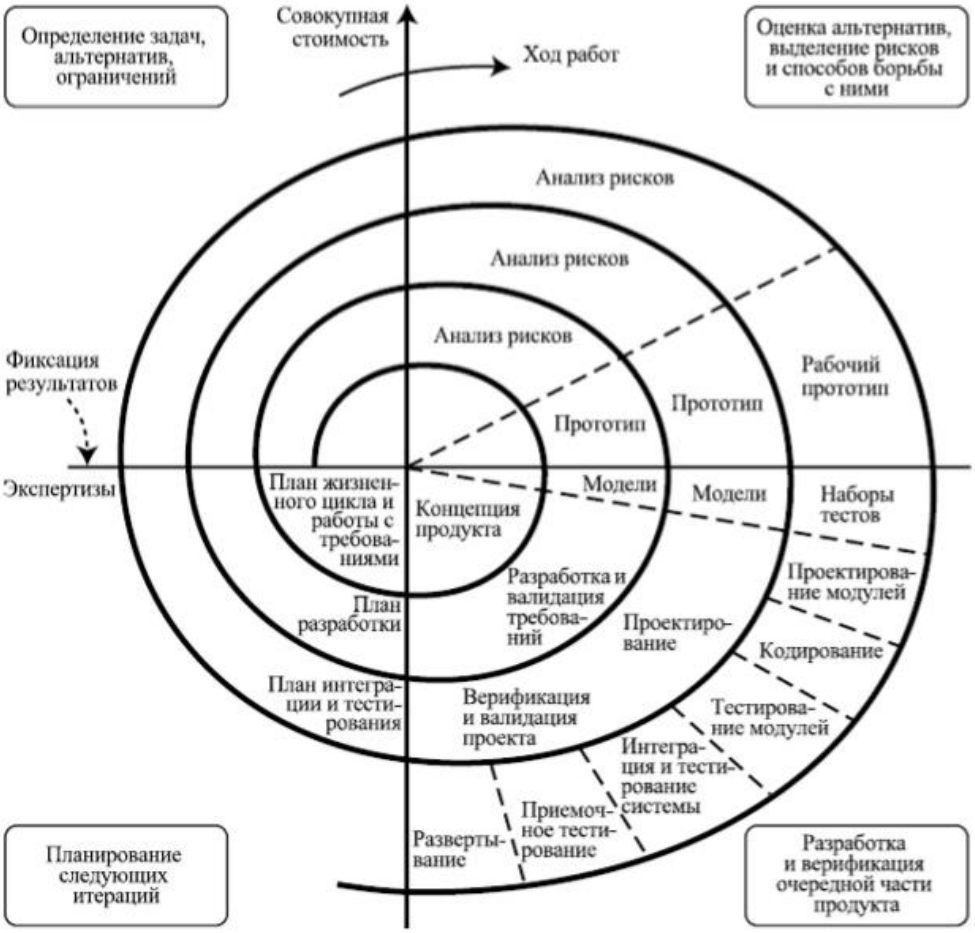


Рис. 1: Спиральная модель

Основная проблема такой модели – определение момента перехода на следующий этап разработки. Чаще всего для её решения вводятся временные ограничения на каждый из этапов жизненного цикла.

Достоинства спиральной модели

1. позволяет быстрее показать пользователям системы работоспособный продукт, тем самым активизируя процесс уточнения и дополнения требований;

2. допускает изменение требований при разработке информационной системы, что характерно для большинства разработок, в том числе и типовых;

3. обеспечивает большую гибкость в управлении проектом;

4. позволяет получить более надежную и устойчивую систему: по мере развития системы ошибки с слабые места обнаруживаются и исправляются на каждой итерации;

5. позволяет совершенствовать процесс разработки: анализ, проводимый в каждой итерации, позволяет проводить оценку того, что должно быть изменено в организации разработки, и улучшить её на следующей итерации;

6. уменьшаются риски заказчика: заказчик может с минимальными для себя финансовыми потерями завершить развитие неперспективного проекта.

Недостатки спиральной модели

1. увеличивается неопределенность у разработчика в перспективах развития проекта (вытекает из 6 пункта достоинств модели);

2. затруднены операции временного и ресурсного планирования всего проекта в целом: требуется ограничивать продолжительность каждой стадии жизненного цикла.

Вывод Спиральная модель имеет гораздо большее количесттво преимуществ по сравнению с другими и отлично подходит для проектов, у которых могут поменяться начальные условия. Такой метод разработки позволяет получить результат за минимальное время, а также вовремя скорректировать направление развития проекта.

ЛАБОРАТОРНАЯ РАБОТА № 6

Тема: Изучение достоинств и недостатков компонентно-ориентированной модели разработки ПО

Цель: изучить достоинства и недостатки компонентно-ориентированной модели разработки ПО.

Компонентно-ориентированная модель

Компонентно-ориентированная модель является развитием спиральной модели и тоже основывается на эволюционной стратегии конструирования. В этой модели конкретизируется содержание этапа конструирования. Оно отражает тот факт, что в современных условиях новая разработка должна основываться на повторном использовании существующих программных компонентов.

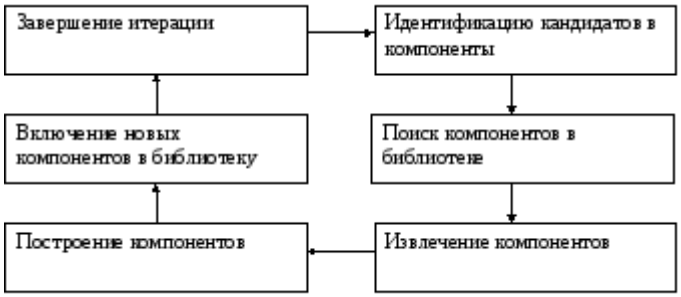


Рисунок 1 – Этапы конструирования компонентно-ориентированной модели

Программные компоненты, созданные в реализованных программных проектах, хранятся в библиотеке. В новом программном проекте, исходя из требований заказчика, выявляются кандидаты в компоненты. Далее проверяется наличие этих кандидатов в библиотеке. Если они найдены, то компоненты извлекаются из библиотеки и используются повторно. В противном случае создаются новые компоненты, которые применяются в проекте и включаются в библиотеку.

Достоинства компонентно-ориентированной модели:

1. уменьшает на 30% время разработки программного продукта;
2. уменьшает стоимость программной разработки до 70%;
3. увеличивает производительность разработки.

Вопросы для самоконтроля по теме:

1. Охарактеризуйте основные особенности современных крупных проектов.
2. Опишите основные этапы разработки программного обеспечения на основе классической модели жизненного цикла.
3. Опишите преимущества и недостатки спиральной модели жизненного цикла программного обеспечения.
4. Охарактеризуйте назначение макетирования.
5. Опишите назначение компонентно-ориентированной модели жизненного цикла программного обеспечения

Тяжеловесные и облегченные процессы

В XXI веке потребности общества в программном обеспечении информационных технологий достигли экстремальных значений. Программная индустрия буквально «захлебывается» от потока самых разнообразных заказов. «Больше процессов разработки,

хороших и разных!» — скандируют заказчики. «Сейчас, сейчас! Только об этом и думаем!» — отвечают разработчики.

Традиционно для упорядочения и ускорения программных разработок предлагались строго упорядочивающие тяжеловесные (heavyweight) процессы. В этих процессах прогнозируется весь объем предстоящих работ, поэтому они называются прогнозирующими (predictive) процессами. Порядок, который должен выполнять при этом человек-разработчик, чрезвычайно строг — «шаг вправо, шаг влево — виртуальный расстрел!». Иными словами, человеческие слабости в расчет не принимаются, а объем необходимой документации способен отнять покой и сон у «совестливого» разработчика.

В последние годы появилась группа новых, облегченных (lightweight) процессов [29]. Теперь их называют подвижными (agile) процессами. Они привлекательны отсутствием бюрократизма, характерного для тяжеловесных (прогнозирующих) процессов. Новые процессы должны воплотить в жизнь разумный компромисс между слишком строгой дисциплиной и полным ее отсутствием. Иначе говоря, порядка в них достаточно для того, чтобы получить разумную отдачу от разработчиков.

Подвижные процессы требуют меньшего объема документации и ориентированы на человека. В них явно указано на необходимость использования природных качеств человеческой натуры (а не на применение действий, направленных наперекор этим качествам).

Более того, подвижные процессы учитывают особенности современного заказчика, а именно частые изменения его требований к программному продукту. Известно, что для прогнозирующих процессов частые изменения требований подобны смерти. В отличие от них, подвижные процессы адаптируют изменения требований и даже выигрывают от этого. Словом, подвижные процессы имеют адаптивную природу.

Таким образом, в современной инфраструктуре программной инженерии существуют два семейства процессов разработки:

- семейство прогнозирующих (тяжеловесных) процессов;
- семейство адаптивных (подвижных, облегченных) процессов.

У каждого семейства есть свои достоинства, недостатки и область применения:

- адаптивный процесс используют при частых изменениях требований, малочисленной группе высококвалифицированных разработчиков и грамотном заказчике, который согласен участвовать в разработке;
- прогнозирующий процесс применяют при фиксированных требованиях и многочисленной группе разработчиков разной квалификации.

Контрольные вопросы

1. Дайте определение технологии конструирования программного обеспечения.
2. Какие этапы классического жизненного цикла вы знаете?
3. Охарактеризуйте содержание этапов классического жизненного цикла.
4. Объясните достоинства и недостатки классического жизненного цикла.
5. Чем отличается классический жизненный цикл от макетирования?
6. Какие существуют формы макетирования?
7. Чем отличаются друг от друга стратегии конструирования ПО?
8. Укажите сходства и различия классического жизненного цикла и инкрементной модели.
9. Объясните достоинства и недостатки инкрементной модели.
10. Чем отличается модель быстрой разработки приложений от инкрементной модели?
11. Объясните достоинства и недостатки модели быстрой разработки приложений.
12. Укажите сходства и различия спиральной модели и классического жизненного

цикла.

13. В чем состоит главная особенность спиральной модели?

14. Чем отличается компонентно-ориентированная модель от спиральной модели и классического жизненного цикла?

15. Перечислите достоинства и недостатки компонентно-ориентированной модели.

16. Чем отличаются тяжеловесные процессы от облегченных процессов?

ЛАБОРАТОРНАЯ РАБОТА №7

Тема: Unified Software Development Process. Тяжеловесные и облегченные процессы.

Цель: изучить Unified Software Development Process. Тяжеловесные и облегченные процессы.

Рациональный унифицированный процесс

Рациональный унифицированный процесс (Rational Unified Process, RUP) является примером так называемого «тяжелого» процесса, детально описанного и предполагающего поддержку собственно разработки исходного кода ПО большим количеством вспомогательных действиям. Примерами подобных действий являются разработка планов, технических заданий, многочисленных проектных моделей, проектной документации и пр. Основная цель такого процесса – отделить успешные практики разработки и сопровождения ПО от конкретных людей, умеющих их применять. Многочисленные вспомогательные действия дают надежду сделать возможным успешное решение задач по конструированию и поддержке сложных систем с помощью имеющихся работников, не обязательно являющихся суперпрофессионалами.

Для достижения этого выполняется иерархическое пошаговое детальное описание предпринимаемых в той или иной ситуации действий, чтобы можно было научить обычного работника действовать аналогичным образом. В ходе проекта создается много промежуточных документов, позволяющих разработчикам последовательно разбивать стоящие перед ними задачи на более простые. Эти же документы служат для проверки правильности решений, принимаемых на каждом шаге, а также отслеживания общего хода работ и уточнения оценок ресурсов, необходимых для получения желаемых результатов.

RUP – одна из спиральных методологий разработки программного обеспечения. Методология поддерживается и развивается компанией Rational Software. В качестве языка моделирования в общей базе знаний используется язык Unified Modelling Language (UML). Итерационная и инкрементная разработка программного обеспечения в RUP предполагает разделение проекта на несколько проектов, которые выполняются последовательно, и каждая итерация разработки четко определена набором целей, которые должны быть достигнуты в конце итерации. Конечная итерация предполагает, что набор целей итерации должен в точности совпадать с набором целей, указанных заказчиком продукта, то есть все требования должны быть выполнены.

Процесс предполагает эволюционирование моделей; итерация цикла разработки однозначно соответствует определенной версии модели программного обеспечения. Каждая из итераций содержит элементы управления жизненным циклом программного обеспечения: анализ и дизайн (моделирование), реализация, интегрирование, тестирование, внедрение. В этом смысле RUP является реализацией спиральной модели, хотя довольно часто изображается в виде графика-таблицы.

В конце каждой итерации (в идеале продолжающейся от 2 до 6 недель) проектная команда должна достичь запланированных на данную итерацию целей, создать или доработать

проектные артефакты и получить промежуточную, но функциональную версию конечного продукта. Итеративная разработка позволяет быстро реагировать на меняющиеся требования, обнаруживать и устранять риски на ранних стадиях проекта, а также эффективно контролировать качество создаваемого продукта.

RUP основан на трех ключевых идеях:

1. Весь ход работ направляется итоговыми целями проекта, выраженными в виде вариантов использования (use cases) – сценариев взаимодействия результирующей программной системы с пользователями или другими системами, при выполнении которых пользователи получают значимые для них результаты и услуги. Разработка начинается с выделения вариантов использования и на каждом шаге контролируется степенью приближения к их реализации.

2. Основным решением, принимаемым в ходе проекта, является архитектура результирующей программной системы. Архитектура устанавливает набор компонентов, из которых будет построено ПО, ответственность каждого из компонентов (т.е. решаемые им подзадачи в рамках общих задач системы), четко определяет интерфейсы, через которые они могут взаимодействовать, а также способы взаимодействия компонентов друг с другом.

3. Архитектура является одновременно основой для получения качественного ПО и базой для планирования работ и оценок проекта в терминах времени и ресурсов, необходимых для достижения определенных результатов. Она оформляется в виде набора графических моделей на языке UML.

Основой процесса разработки являются планируемые и управляемые итерации, объем которых (реализуемая в рамках итерации функциональность и набор компонентов) определяется на основе архитектуры.

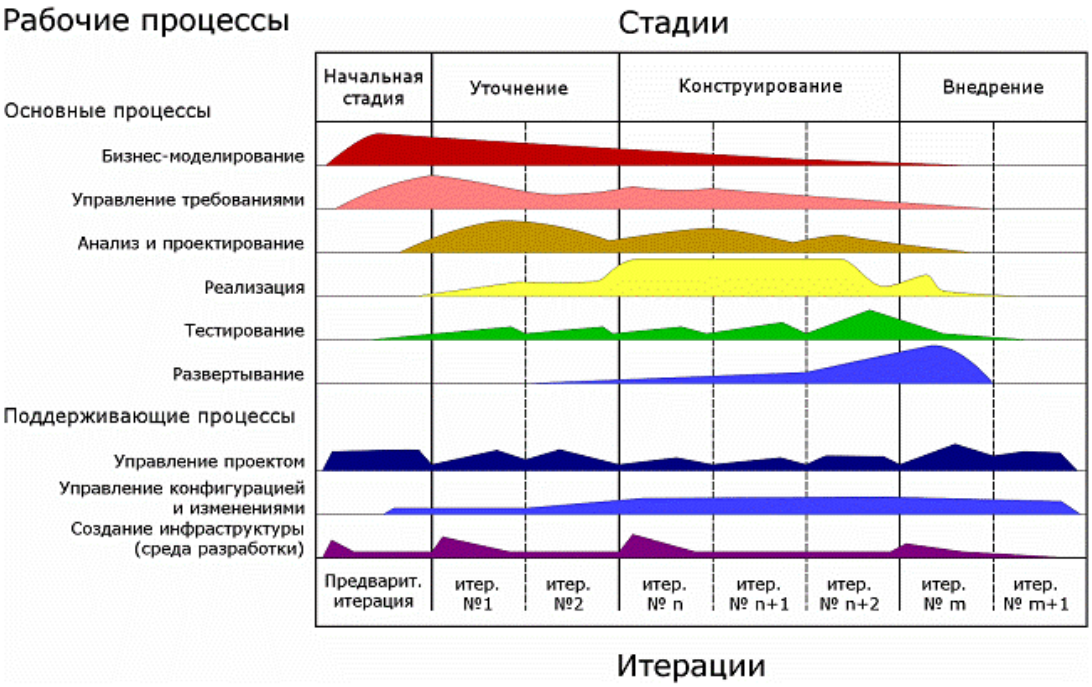


Рис. Структура рационального унифицированного процесса (RUP)

На данном рисунке представлены два измерения: горизонтальная ось представляет время и показывает временные аспекты жизненного цикла процесса; вертикальная ось представляет дисциплины, которые определяют физическую структуру процесса. На рисунке видно, как с течением времени изменяются акценты в проекте. Например, в

ранних итерациях больше времени отводится требованиям; в поздних итерациях больше времени отводится реализации. Горизонтальная ось сформирована из временных отрезков – итераций, каждая из которых является самостоятельным циклом разработки; цель цикла – принести в конечной продукт некоторую заранее определенную осязаемую доработку, полезную с точки зрения заинтересованных лиц.

Структура рационального унифицированного процесса (RUP). По оси времени жизненный цикл делится на четыре основные фазы.

1. Начало (Inception) – формирование концепции проекта, понимание того, что мы создаем, представление о продукте (vision), разработка бизнес-плана (business case), подготовка прототипа программы или частичного решения. Это фаза сбора информации и анализа требований, определение образа проекта в целом. Цель – получить поддержку и финансирование. В конечной итерации результат этого этапа – техническое задание.

2. Проектирование, разработка (Elaboration) – уточнение плана, понимание того, как мы это создаем, проектирование, планирование необходимых действий и ресурсов, детализация особенностей. Завершается этап исполняемой архитектурой, когда все архитектурные решения приняты и риски учтены. Исполняемая архитектура представляет собой работающее программное обеспечение, которое демонстрирует реализацию основных архитектурных решений. В конечной итерации это – технический проект.

3. Реализация, создание системы (Construction) – этап расширения функциональности системы, заложенной в архитектуре. В конечной итерации это – рабочий проект.

4. Внедрение, развертывание (Transition). Создание конечной версии продукта. Фаза внедрения продукта, поставка продукта конкретному пользователю (тиражирование, доставка и обучение).

Вертикальная ось состоит из дисциплин, каждая из них может быть более детально расписана с точки зрения выполняемых задач, ответственных за них ролей, продуктов, которые подаются задачам на вход и выпускаются в ходе их выполнения и т.д.

По этой оси располагаются ключевые дисциплины жизненного цикла RUP, которые часто на русском языке называют процессами, хотя это не совсем верно с точки зрения данной методологии, поддерживаемые инструментальными средствами IBM (и/или третьих фирм).

- Бизнес-анализ и моделирование (Business modeling) обеспечивает реализацию принципов моделирования с целью изучения бизнеса организации и накопления знаний о нем, оптимизации бизнес-процессов и принятия решения об их частичной или полной автоматизации.
- Управление требованиями (Requirements) посвящено получению информации от заинтересованных лиц и ее преобразованию в набор требований, определяющих содержание разрабатываемой системы и подробно описывающих ожидания от того, что система должна делать.
- Анализ и проектирование (Analysis and design) охватывает процедуры преобразования требований в промежуточные описания (модели), представляющие, как эти требования должны быть реализованы.
- Реализация (Implementation) охватывает разработку кода, тестирование на уровне разработчиков и интеграцию компонентов, подсистем и всей системы в соответствии с установленными спецификациями.
- Тестирование (Test) посвящено оценке качества создаваемого продукта.
- Развертывание (Deployment) охватывает операции, имеющие место при передаче продуктов заказчикам и обеспечении доступности продукта конечным пользователям.

- Конфигурационное управление и управление изменениями (Configuration management) посвящено синхронизации промежуточных и конечных продуктов и управлению их развитием в ходе проекта и поиском скрытых проблем.
- Управление проектом (Management) посвящено планированию проекта, управлению рисками, контролю хода его выполнения и непрерывной оценке ключевых показателей.
- Управление средой (Environment) включает элементы формирования среды разработки информационной системы и поддержки проектной деятельности.

В зависимости от специфики проекта могут быть использованы любые средства IBM Rational, а также третьих фирм. В RUP рекомендовано следовать шести практикам, позволяющим успешно разрабатывать проект: итеративная разработка; управление требованиями; использование модульных архитектур; визуальное моделирование; проверка качества; отслеживание изменений.

Неотъемлемую часть RUP составляют артефакты (artefact), прецеденты (precedent) и роли (role). Артефакты – это некоторые продукты проекта, порождаемые или используемые в нем при работе над окончательным продуктом. Прецеденты – это последовательности действий, выполняемых системой для получения наблюдаемого результата. Фактически любой результат работы индивидуума или группы является артефактом, будь то документ анализа, элемент модели, файл кода, тестовый скрипт, описание ошибки и т.п. За создание того или иного вида артефактов отвечают определенные специалисты. Таким образом, RUP четко определяет обязанности – роли – каждого члена группы разработки на том или ином этапе, то есть когда и кто должен создать тот или иной артефакт. Весь процесс разработки программной системы рассматривается в RUP как процесс создания артефактов – начиная с первоначальных документов анализа и заканчивая исполняемыми модулями, руководствами пользователя и т.п.

Для компьютерной поддержки процессов RUP в IBM разработан широкий набор инструментальных средств:

- Rational Rose – CASE-средство визуального моделирования информационных систем, имеющее возможности генерирования элементов кода. Специальная редакция продукта – Rational Rose RealTime – позволяет на выходе получить исполняемый модуль;
- Rational Requisite Pro – средство управления требованиями, которое позволяет создавать, структурировать, устанавливать приоритеты, отслеживать, контролировать изменения требований, возникающие на любом этапе разработки компонентов приложения;
- Rational ClearQuest – продукт для управления изменениями и отслеживания дефектов в проекте (bug tracking), тесно интегрирующийся со средствами тестирования и управления требованиями и представляющий собой единую среду для связывания всех ошибок и документов между собой;
- Rational SoDA – продукт для автоматического генерирования проектной документации, позволяющий установить корпоративный стандарт на внутрифирменные документы. Возможно также приведение документации к уже существующим стандартам (ISO, CMM);
- Rational Purify, Rational Quantify Rational PureCoverage, – средства тестирования и отладки;
- Rational Visual Quantify – средство измерения характеристик для разработчиков приложений и компонентов, программирующих на C/C++, Visual Basic и Java; помогает определять и устранять узкие места в производительности ПО;
- Rational Visual PureCoverage – автоматически определяет области кода, которые не подвергаются тестированию;
- Rational ClearCase – продукт для управления конфигурацией программ (Rational's Software Configuration Management, SCM), позволяющий производить версионный

контроль всех документов проекта. С его помощью можно поддерживать несколько версий проектов одновременно, быстро переключаясь между ними. Rational Requisite Pro поддерживает обновления и отслеживает изменения в требованиях для группы разработчиков;
- SQA TeamTest – средство автоматизации тестирования;
- Rational TestManager – система управления тестированием, которая объединяет все связанные с тестированием инструментальные средства, артефакты, сценарии и данные;
- Rational Robot – инструмент для создания, модификации и автоматического запуска тестов;
- SiteLoad, SiteCheck – средства тестирования Web-сайтов на производительность и наличие неработающих ссылок;
- Rational PerformanceStudio – измерение и предсказание характеристик производительности систем.

Этот набор продуктов постоянно совершенствуется и пополняется. Так, например, недавний продукт IBM Rational Software Architect (RSA) является частью IBM Software Development Platform – набора инструментов, поддерживающих жизненный цикл разработки программных систем. Продукт IBM Rational Software Architect предназначен для построения моделей разрабатываемых программных систем с использованием унифицированного языка моделирования UML 2.0, прежде всего моделей архитектуры разрабатываемого приложения. Тем не менее, RSA объединяет в себе функции таких программных продуктов, как Rational Application Developer, Rational Web Developer и Rational Software Modeler, тем самым предоставляя возможность архитекторам и аналитикам создавать различные представления разрабатываемой информационной системы с использованием языка UML 2.0, а разработчикам – выполнять разработку J2EE, XML, веб-сервисов и т.д.

Следуя принципам RUP, Rational Software Architect позволяет создавать необходимые модели в рамках рабочих процессов таких дисциплин, как:

- бизнес-анализ и моделирование (Business modeling);
- управление требованиями (Requirements);
- анализ и проектирование (Analysis and Design);
- реализация (Implementation).

Кроме того, Rational Software Architect поддерживает технологию разработки, управляемой моделями (model-driven development, MDD), которая позволяет моделировать программное обеспечение на различных уровнях абстракции с возможностью трассируемости.

Экстремальное программирование

Экстремальное программирование (Extreme Programming, XP) возникло как эволюционный метод разработки ПО «снизу-вверх». Этот подход является примером так называемого метода гибкой («живой») разработки (Agile Development Method). В группу гибких методов входят, помимо экстремального программирования, методы SCRUM, DSDM (Dynamic Systems Development Method, метод разработки динамичных систем), Feature-Driven Development (разработка, управляемая функциями системы) и др.

Основные принципы гибкой разработки ПО зафиксированы в манифесте гибкой разработки, появившемся в 2000 году.

1. Люди, участвующие в проекте, и их общение более важны, чем процессы и инструменты.
2. Работающая программа более важна, чем исчерпывающая документация.

3. Сотрудничество с заказчиком более важно, чем обсуждение деталей контракта.
4. Отработка изменений более важна, чем следование планам.

Гибкие методы появились как протест против чрезмерной бюрократизации разработки ПО, обилия побочных, не являющихся необходимыми для получения конечного результата документов, которые приходится оформлять при проведении проекта в соответствии с большинством «тяжелых» процессов, дополнительной работы по поддержке фиксированного процесса организации, как это требуется в рамках, например, CMM. Большая часть таких работ и документов не имеет прямого отношения к разработке ПО и обеспечению его качества, а предназначена для соблюдения формальных пунктов контрактов на разработку, получения и подтверждения сертификатов на соответствие различным стандартам.

Гибкие методы позволяют большую часть усилий разработчиков сосредоточить собственно на задачах разработки и удовлетворения реальных потребностей пользователей. Отсутствие кипы документов и необходимости поддерживать их в связном состоянии позволяет более быстро и качественно реагировать на изменения в требованиях и в окружении, в котором придется работать будущей программе

ЛАБОРАТОРНАЯ РАБОТА № 8

Тема: Экстремальное программирование. Парное программирование: преимущества и недостатки.

Цель: изучить основы экстремального программирования

Экстремальное программирование (eXtreme Programming, XP) — облегченный (подвижный) процесс (или методология), главный автор которого — Кент Бек (1999). XP-процесс ориентирован на группы малого и среднего размера, строящие программное обеспечение в условиях неопределенных или быстро изменяющихся требований. XP-группу образуют до 10 сотрудников, которые размещаются в одном помещении. Основная идея XP — устранить высокую стоимость изменения, характерную для приложений с использованием объектов, паттернов* и реляционных баз данных. Поэтому XP-процесс должен быть высокодинамичным процессом. XP-группа имеет дело с изменениями требований на всем протяжении итерационного цикла разработки, причем цикл состоит из очень коротких итераций. Четырьмя базовыми действиями в XP-цикле являются: кодирование, тестирование, выслушивание заказчика и проектирование. Динамизм обеспечивается с помощью четырех характеристик: непрерывной связи с заказчиком (и в пределах группы), простоты (всегда выбирается минимальное решение), быстрой обратной связи (с помощью модульного и функционального тестирования), смелости в проведении профилактики возможных проблем.

* Паттерн является решением типичной проблемы в определенном контексте. Большинство принципов, поддерживаемых в XP (минимальность, простота, эволюционный цикл разработки, малая длительность итерации, участие пользователя, оптимальные стандарты кодирования и т. д.), продиктованы здравым смыслом и применяются в любом упорядоченном процессе. Просто в XP эти принципы, как показано в табл. 1, достигают «экстремальных значений».

Таблица 1. Экстремумы в экстремальном программировании

Практика здравого смысла	XP-экстремум	XP-реализация
Проверки кода	Код проверяется все время	Парное программирование
Тестирование	Тестирование выполняется все время, даже с помощью заказчиков	Тестирование модуля, функциональное тестирование
Проектирование	Проектирование является частью ежедневной деятельности каждого разработчика	Реорганизация (refactoring)
Простота	Для системы выбирается простейшее проектное решение, поддерживающее ее текущую функциональность	Самая простая вещь, которая могла бы работать
Архитектура	Каждый постоянно работает над уточнением архитектуры	Метафора
Тестирование интеграции	Интегрируется и тестируется несколько раз в день	Непрерывная интеграция
Короткие итерации	Итерации являются предельно короткими, продолжаются секунды, минуты, часы, а не недели, месяцы или годы	Игра планирования

Тот, кто принимает принцип «минимального решения» за хакерство, ошибается, в действительности XP — строго упорядоченный процесс. Простые решения, имеющие высший приоритет, в настоящее время рассматриваются как наиболее ценные части системы, в отличие от проектных решений, которые пока не нужны, а могут (в условиях изменения требований и операционной среды) и вообще не понадобиться.

Базис XP образуют перечисленные ниже двенадцать методов.

1. Игра планирования (Planning game) — быстрое определение области действия следующей реализации путем объединения деловых приоритетов и технических оценок. Заказчик формирует область действия, приоритетность и сроки с точки зрения бизнеса, а разработчики оценивают и прослеживают продвижение (прогресс)

2. Частая смена версий (Small releases) — быстрый запуск в производство простой системы. Новые версии реализуются в очень коротком (двухнедельном) цикле.

3. Метафора (Metaphor) — вся разработка проводится на основе простой, общедоступной истории о том, как работает вся система.

4. Простое проектирование (Simple design) — проектирование выполняется настолько просто, насколько это возможно в данный момент.

5. Тестирование (Testing) — непрерывное написание тестов для модулей, которые должны выполняться безупречно; заказчики пишут тесты для демонстрации законченности функций. «Тестируй, а затем кодируй» означает, что входным критерием для написания кода является «отказавший» тестовый вариант.

6. Реорганизация (Refactoring) — система реструктурируется, но ее поведение не изменяется; цель — устранить дублирование, улучшить взаимодействие, упростить систему или добавить в нее гибкость.

7. Парное программирование (Pair programming) — весь код пишется двумя программистами, работающими на одном компьютере.

8. Коллективное владение кодом (Collective ownership) — любой разработчик может улучшать любой код системы в любое время.

9. Непрерывная интеграция (Continuous integration) — система интегрируется и строится много раз в день, по мере завершения каждой задачи. Непрерывное регрессионное тестирование, то есть повторение предыдущих тестов, гарантирует, что изменения требований не приведут к регрессу функциональности.

10. 40-часовая неделя (40-hour week) — как правило, работают не более 40 часов в

неделю. Нельзя удваивать рабочую неделю за счет сверхурочных работ.

11. Локальный заказчик (On-site customer) — в группе все время должен находиться представитель заказчика, действительно готовый отвечать на вопросы разработчиков.

12. Стандарты кодирования (Coding standards) — должны выдерживаться правила, обеспечивающие одинаковое представление программного кода во всех частях программной системы.

Игра планирования и частая смена версий зависят от заказчика, обеспечивающего набор «историй» (коротких описаний), характеризующих работу, которая будет выполняться для каждой версии системы. Версии генерируются каждые две недели, поэтому разработчики и заказчик должны прийти к соглашению о том, какие истории будут осуществлены в пределах двух недель. Полную функциональность, требуемую заказчику, характеризует пул историй; но для следующей двухнедельной итерации из пула выбирается подмножество историй, наиболее важное для заказчика. В любое время в пул могут быть добавлены новые истории, таким образом, требования могут быстро изменяться. Однако процессы двухнедельной генерации основаны на наиболее важных функциях, входящих в текущий пул, следовательно, изменчивость управляется. Локальный заказчик обеспечивает поддержку этого стиля итерационной разработки.

«Метафора» обеспечивает глобальное «видение» проекта. Она могла бы рассматриваться как высокоуровневая архитектура, но XP подчеркивает желательность проектирования при минимизации проектной документации. Точнее говоря, XP предлагает непрерывное перепроектирование (с помощью реорганизации), при котором нет нужды в детализированной проектной документации, а для инженеров сопровождения единственным надежным источником информации является программный код. Обычно после написания кода проектная документация выбрасывается. Проектная документация сохраняется только в том случае, когда заказчик временно теряет способность придумывать новые истории. Тогда систему помещают в «нафталин» и пишут руководство страниц на пять-десять по «нафталиновому» варианту системы. Использование реорганизации приводит к реализации простейшего решения, удовлетворяющего текущую потребность. Изменения в требованиях заставляют отказываться от всех «общих решений».

Парное программирование — один из наиболее спорных методов в XP, оно влияет на ресурсы, что важно для менеджеров, решающих, будет ли проект использовать XP. Может показаться, что парное программирование удваивает ресурсы, но исследования доказали: парное программирование приводит к повышению качества и уменьшению времени цикла. Для согласованной группы затраты увеличиваются на 15%, а время цикла сокращается на 40-50%. Для Интернет-среды увеличение скорости продаж покрывает повышение затрат. Сотрудничество улучшает процесс решения проблем, улучшение качества существенно снижает затраты сопровождения, которые превышают стоимость дополнительных ресурсов по всему циклу разработки.

Коллективное владение означает, что любой разработчик может изменять любой фрагмент кода системы в любое время. Непрерывная интеграция, непрерывное регрессионное тестирование и парное программирование XP обеспечивают защиту от возникающих при этом проблем.

«Тестируй, а затем кодируй» — эта фраза выражает акцент XP на тестировании. Она отражает принцип, по которому сначала планируется тестирование, а тестовые варианты разрабатываются параллельно анализу требований, хотя традиционный подход состоит в тестировании «черного ящика». Размышление о тестировании в начале цикла жизни — хорошо известная практика конструирования ПО (правда, редко осуществляемая практически).

Основным средством управления XP является метрика, а среда метрик — «большая визуальная диаграмма». Обычно используют 3-4 метрики, причем такие, которые видимы всей группе. Рекомендуемой в XP метрикой является «скорость проекта» — количество историй заданного размера, которые могут быть реализованы в итерации.

При принятии XP рекомендуется осваивать его методы по одному, каждый раз выбирая метод, ориентированный на самую трудную проблему группы. Конечно, все эти методы являются «не более чем правилами» — группа может в любой момент поменять их (если ее сотрудники достигли принципиального соглашения по поводу внесенных изменений). Защитники XP признают, что XP оказывает сильное социальное воздействие, и не каждый может принять его. Вместе с тем, XP — это методология, обеспечивающая преимущества только при использовании законченного набора базовых методов.

Рассмотрим структуру «идеального» XP-процесса. Основным структурным элементом процесса является XP-реализация, в которую многократно вкладывается базовый элемент — XP-итерация. В состав XP-реализации и XP-итерации входят три фазы — исследование, блокировка, регулирование. Исследование (exploration) — это поиск новых требований (историй, задач), которые должна выполнять система. Блокировка (commitment) — выбор для реализации конкретного подмножества из всех возможных требований (иными словами, планирование). Регулирование (steering) — проведение разработки, воплощение плана в жизнь.

XP рекомендует: первая реализация должна иметь длительность 2-6 месяцев, продолжительность остальных реализаций — около двух месяцев, каждая итерация длится приблизительно две недели, а численность группы разработчиков не превышает 10 человек. XP-процесс для проекта с семью реализациями, осуществляемый за 15 месяцев, показан на рис. 1.

Процесс инициируется начальной исследовательской фазой.

Фаза исследования, с которой начинается любая реализация и итерация, имеет клапан «пропуска», на этой фазе принимается решение о целесообразности дальнейшего продолжения работы.

Предполагается, что длительность первой реализации составляет 3 месяца, длительность второй — седьмой реализаций — 2 месяца. Вторая — седьмая реализации образуют период сопровождения, характеризующий природу XP-проекта. Каждая итерация длится две недели, за исключением тех, которые относят к поздней стадии реализации — «запуску в производство» (в это время темп итерации ускоряется).

Наиболее трудна первая реализация — пройти за три месяца от обычного старта (скажем, отдельный сотрудник не зафиксировал никаких требований, не определены ограничения) к поставке заказчику системы промышленного качества очень сложно

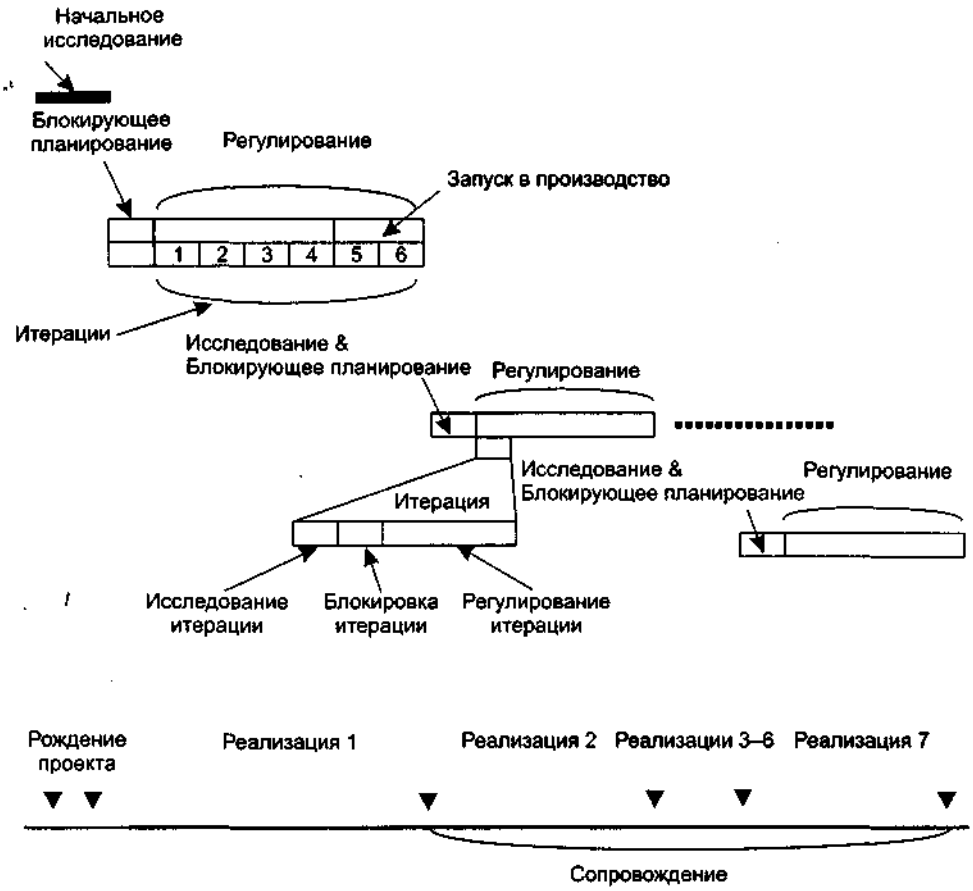


Рис. . Идеальный XP-процесс

ЛАБОРАТОРНАЯ РАБОТА № 9

Тема : Стандарты. Характеристики качества ПО

Цель: Знакомство с ГОСТ 28.195-89 «Оценка качества программных средств. Общие положения». Определить способы получения информации о ПС. Формирование информационно - правовых компетенции обучающихся.

Качество программных средств

Введение

Одной из важнейших проблем обеспечения качества программных средств является формализация характеристик качества и методология их оценки. Для определения адекватности качества функционирования, наличия технических возможностей программных средств к взаимодействию, совершенствованию и развитию необходимо использовать стандарты в области оценки характеристик их качества.

Показатели качества программного обеспечения устанавливают ГОСТ 28.195-89 «Оценка качества программных средств. Общие положения» и ГОСТ Р ИСО/МЭК 9126 «Информационная технология. Оценка программной продукции. Характеристика качества и руководства по их применению». Одновременное существование двух действующих стандартов, нормирующих одни и те же показатели, ставит вопрос об их гармонизации. Ниже рассмотрим каждый из перечисленных стандартов.

ГОСТ 28.195-89 «Оценка качества программных средств. Общие положения» устанавливает общие положения по оценке качества программных средств, номенклатуру и применяемость показателей качества.

Оценка качества ПС представляет собой совокупность операций, включающих выбор номенклатуры показателей качества оцениваемого ПС, определение значений этих показателей и сравнение их с базовыми значениями.

Методы определения показателей качества ПС различаются:

- по способам получения информации о ПС – измерительный, регистрационный, органолептический, расчетный;
- по источникам получения информации – экспертный, социологический.

Измерительный метод основан на получении информации о свойствах и характеристиках ПС с использованием инструментальных средств. Например, с использованием этого метода определяется объем ПС - число строк исходного текста программ и число строк - комментариев, число операторов и операндов, число исполненных операторов, число ветвей в программе, число точек входа (выхода), время выполнения ветви программы, время реакции и другие показатели.

Регистрационный метод основан на получении информации во время испытаний или функционирования ПС, когда регистрируются и подсчитываются определенные события, например, время и число сбоев и отказов, время передачи управления другим модулям, время начала и окончания работы.

Органолептический метод основан на использовании информации, получаемой в результате анализа восприятия органов чувств (зрения, слуха), и применяется для определения таких показателей как удобство применения, эффективность и т. п.

Расчетный метод основан на использовании теоретических и эмпирических зависимостей (на ранних этапах разработки), статистических данных, накапливаемых при испытаниях, эксплуатации и сопровождении ПС. При помощи расчетного метода определяются длительность и точность вычислений, время реакции, необходимые ресурсы.

Определение значений показателей качества ПС *экспертным методом* осуществляется группой экспертов-специалистов, компетентных в решении данной задачи, на базе их опыта и интуиции. Экспертный метод применяется в случаях, когда задача не может быть решена никаким другим из существующих способов или другие способы являются значительно более трудоемкими. Экспертный метод рекомендуется применять при определении показателей наглядности, полноты и доступности программной документации, легкости освоения, структурности.

Социологические методы основаны на обработке специальных анкет-вопросников.

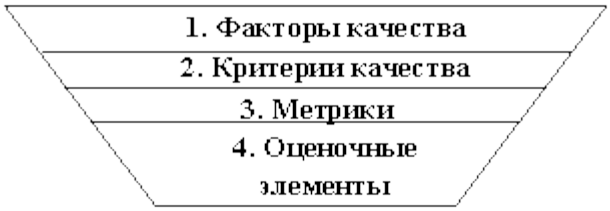


Рис. 1 – Уровни системы показателей качества

Показатели качества объединены в систему из четырех уровней. Каждый вышестоящий уровень содержит в качестве составляющих показатели нижестоящих уровней (рисунок 1).

Стандарт ИСО 9126 (ГОСТ Р ИСО/МЭК 9126) «Информационная технология. Оценка программной продукции. Характеристика качества и руководства по их применению».

Определенные настоящим стандартом характеристики дополнены рядом требований по выбору метрик и их измерению для различных проектов ПС. Они применимы к любому типу ПС, включая компьютерные программы и данные, содержащиеся в программируемом оборудовании. Эти характеристики обеспечивают согласованную терминологию для анализа качества ПС. Кроме того, они определяют схему для выбора и специфицирования требований к качеству ПС, а также для

сопоставления возможностей различных программных продуктов, таких как функциональные возможности, надежность, практичность и эффективность.

Все множество атрибутов качества ПС может быть классифицировано в структуру иерархического дерева характеристик и субхарактеристик. Самый высший уровень этой структуры состоит из характеристик качества, а самый нижний уровень – из их атрибутов. Эта иерархия не строгая, поскольку некоторые атрибуты могут быть связаны с более чем одной субхарактеристикой. Таким же образом, внешние свойства (такие, как пригодность, корректность, устойчивость к ошибкам или временная эффективность) влияют на наблюдаемое качество. Недостаток качества в использовании (например, пользователь не может закончить задачу) может быть прослежен к внешнему качеству (например, функциональная пригодность или простота использования) и связанным с ним внутренним атрибутам, которые необходимо изменить.

Внутренние метрики могут применяться в ходе проектирования и программирования к неисполняемым компонентам ПС (таким, как спецификация или исходный программный текст). При разработке ПС промежуточные продукты следует оценивать с использованием внутренних метрик, которые измеряют свойства программ, и могут быть выведены из моделируемого поведения. Основная цель внутренних метрик – обеспечивать, чтобы было достигнуто требуемое внешнее качество. Внутренние метрики дают возможность пользователям, испытателям и разработчикам оценивать качество ЖЦ программ и заниматься вопросами технологического обеспечения качества задолго до того, как ПС становится готовым исполняемым продуктом.

Внутренние метрики позволяют измерять внутренние атрибуты или формировать признаки внешних атрибутов путем анализа статических свойств промежуточных или поставляемых программных компонентов. Измерения внутренних метрик используют категории, числа или характеристики элементов из состава ПС, которые, например, имеются в процедурах исходного программного текста, в графе потока управления, в потоке данных и в представлениях изменения состояний памяти. Документация также может оцениваться с использованием внутренних метрик.

Внешние метрики используют меры ПС, выведенные из поведения системы, частью которых они являются, путем испытаний, эксплуатации или наблюдения исполняемого ПС или системы. Перед приобретением или использованием ПС его следует оценить с использованием метрик, основанных на деловых и профессиональных целях, связанных с использованием, эксплуатацией и управлением продуктом в определенной организационной и технической среде. Внешние метрики обеспечивают заказчикам, пользователям, испытателям и разработчикам возможность определять качество ПС в ходе испытаний или эксплуатации.

Когда требования к качеству ПС определены, в них должны быть перечислены характеристики и субхарактеристики, которые составляют полный набор показателей качества. Затем определяются подходящие внешние метрики и их приемлемые диапазоны значений, устанавливающие количественные и качественные критерии, которые подтверждают, что ПС удовлетворяет потребностям заказчика и пользователя. Далее определяются и специфицируются внутренние атрибуты качества, чтобы спланировать удовлетворение требуемых внешних характеристик качества в конечном продукте и обеспечивать их в промежуточных продуктах в ходе разработки. Подходящие внутренние метрики и приемлемые диапазоны специфицируются для получения числовых значений или категорий внутренних характеристик качества, чтобы их можно было использовать для проверки того, что промежуточные продукты в процессе разработки удовлетворяют внутренним спецификациям качества. Рекомендуется использовать внутренние метрики,

которые имеют наиболее сильные связи с целевыми внешними метриками, чтобы они могли помогать при прогнозировании значений внешних метрик.

Метрики качества в использовании измеряют, в какой степени продукт удовлетворяет потребности конкретных пользователей в достижении заданных целей с результативностью, продуктивностью и удовлетворением в заданном контексте использования. При этом результативность подразумевает точность и полноту достижения определенных целей пользователями при применении ПС; продуктивность соответствует соотношению израсходованных ресурсов и результативности при эксплуатации ПС, а удовлетворенность – психологическое отношение к качеству использования продукта. Эта метрика не входит в число шести базовых характеристик ПС, регламентируемых стандартом ИСО 9126, однако рекомендуется для интегральной оценки результатов функционирования комплексов программ.

Оценивание качества в использовании должно подтверждать его для определенных сценариев и задач, оно составляет полный объединенный эффект характеристик качества ПС для пользователя. *Качество в использовании* – это восприятие пользователем качества системы, содержащей ПС, и оно измеряется скорее в терминах результатов использования комплекса программ, чем собственных внутренних свойств ПС. Связь качества в использовании с другими характеристиками качества ПС зависит от типа пользователя, так, например, для конечного пользователя качество в использовании обусловливают, в основном, характеристики функциональных возможностей, надежности, практичности и эффективности, а для персонала сопровождения ПС качество в использовании определяет сопровождаемость. На качество в использовании могут влиять любые характеристики качества, и это понятие шире, чем практичность, которая связана с простотой использования и привлекательностью. Качество в использовании, в той или иной степени, характеризуется сложностью применения комплекса программ, которую можно описать трудоемкостью использования с требуемой результативностью. Многие характеристики и субхарактеристики ПС обобщенно отражаются неявными технико-экономическими показателями, которые поддерживают функциональную пригодность конкретного ПС. Однако их измерение и оценка влияния на показатели качества, представляет сложную проблему.

Ход работы:

Задание 1. Провести сравнение понятий «качество» государственным и международным стандартами. Выписать документы, в которых даны данные определения.

Задание 2. Опишите методы получения информации о ПС по ГОСТу. Для каждого метода выделите источник информации.

Задание 3. Выберите стандарты для оценки качества ПС. Перечислите критерии надежности ПС по ГОСТу.

ЛАБОРАТОРНАЯ РАБОТА № 10

Тема : Модели качества процессов разработки. Модель зрелости.
Цель: изучить модели качества процессов разработки и модели зрелости

Оценка качества процесса разработки

Существует два подхода, которые могут применяться для оценки качества программного продукта.

- Оценить качество конечного продукта.

- Оценить качество процесса разработки.

Оценить качество конечного продукта можно тестированием и эксплуатацией. На это должно быть отведено время после завершения основной работы над программой. А вот второй подход должен стать частью долговременной стратегии компании.

В модели определено пять уровней зрелости организации

- Начальный уровень. На этом уровне процесс разработки характеризуется практическим отсутствием процессов управления. Успех проекта зависит от индивидуальных усилий, личных качеств и даже героизма участников проекта.
- Повторяемый уровень. На этом уровне зрелости в компании должны быть внедрены основные процессы управления для отслеживания стоимости, графика проекта и его функциональности. Уровень характеризуется тем, что управление проектами основывается на накопленном опыте и ранее достигнутые успехи будут повторены на подобных приложениях.
- Определенный уровень. Процесс разработки программного обеспечения (как на уровне управленческой, так и инженерной деятельности) документирован, стандартизован и интегрирован на уровне всей организации. Процесс перестает зависеть от индивидуальных качеств отдельных разработчиков и не может скатиться на более низкие уровни в кризисных ситуациях.
- Управляемый уровень. В компании устанавливаются детальные количественные показатели на процесс разработки и качество продукта. И процесс разработки, и продукты - понимаемы и контролируемы.
- Оптимизирующий уровень. Продолжающееся совершенствование процесса разработки на основе анализа текущих результатов процесса и применения инновационных идей и технологий.

Определение возможностей и улучшение процесса создания программного обеспечения

Данная модель очень близка к модели зрелости, но уровни возможностей могут быть применены не только к организации в целом, но и к отдельно взятым процессам. Модели такого типа часто называют непрерывными. В непрерывных моделях один процесс может находиться на низком уровне зрелости, а другой - на высоком уровне. Стандарт определяет шесть уровней зрелости процесса.

- Уровень 0. Процесс не выполняется.
- Уровень 1. Выполняемый процесс.
- Уровень 2. Управляемый процесс.
- Уровень 3. Установленный процесс.
- Уровень 4. Предсказуемый процесс.
- Уровень 5. Оптимизирующий процесс.

Во время оценки и улучшения качества процессов выполняются задачи , описанные ниже.

- Сравнение процесса разработки программного обеспечения, существующего в данной организации, с описанной в стандарте моделью. Анализ результатов дает возможность определить сильные и слабые стороны процесса, его внутренние риски.

- Оценка возможности улучшения данного процесса на основе определения текущих возможностей.
- Техническая реализация поставленных задач на основе сформулированных целей совершенствования процесса. После этого весь цикл работ начинается сначала

ЛАБОРАТОРНАЯ РАБОТА № 11

Тема : . Диаграммы переходов состояний.
Архитектура программного обеспечения. Микро - и макроахитектура.
Архитектурные стили и шаблоны проектирования. Метод SADT

Цель: разработать диаграммы переходов состояний по предложенной информационной системе.

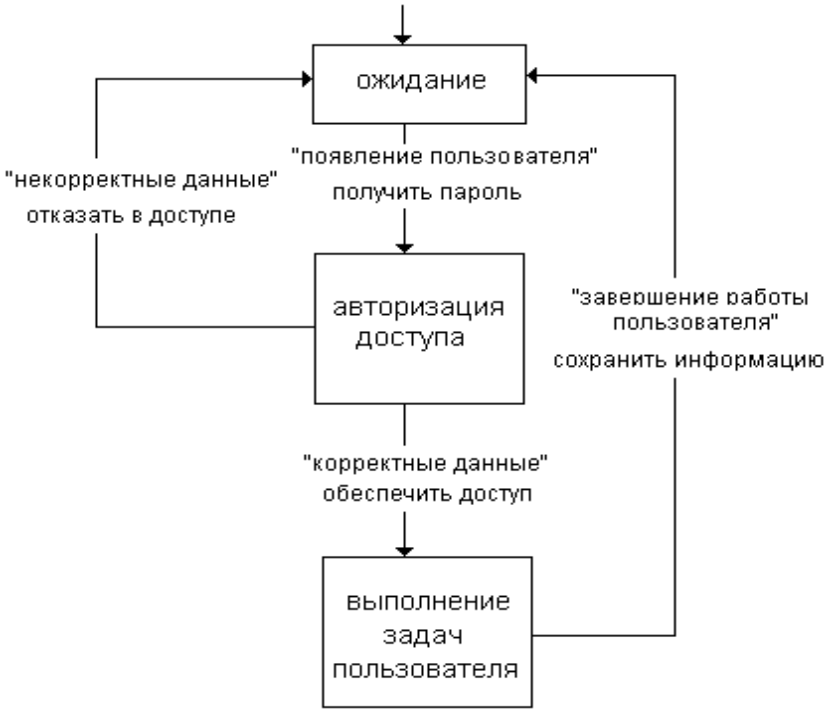
Исходные данные (задание):

1. Ознакомиться с теоретическим материалом по составлению диаграмм переходов состояний.
2. Рассмотреть предложенные диаграммы переходов состояний.
3. Сформировать диаграммы переходов состояний по предложенной информационной системе.
4. Оформить отчет по работе.

Диаграммы переходов состояний (std)
Используются для моделирования поведения систе **STD** (***State Transition Diagrams***)мы, зависящего от времени или реакций системы на некоторые события.

STD состоит из следующих объектов:

- **Состояние** – моделируемая система в любой заданный момент времени должна находится точно в одном из конечного множества состояний
- **Начальное состояние** является стартовой точкой для начального системного перехода, соответствующего состоянию системы после её инсталляции. STD должна иметь только одно начальное состояние, а также любое (конечное) число завершающих состояний.
- **Переход** определяет перемещение моделируемой системы из одного состояния в другое. При этом имя перехода идентифицирует событие, являющееся причиной перехода и управляющее им. Это событие обычно состоит из управляющего потока (сигнала), возникающего как во внешнем мире, так: и внутри системы при выполнении некоторого условия.
- **Действие** – это операция, которая может быть связана с переходом, и выполняющаяся при выполнении перехода



На STD состояния представляются узлами, а переходы - дугами Условия идентифицируются именем перехода и возбуждают выполнение перехода. Действия или отклики на события привязываются к переходам и записываются под соответствующим условием. Начальное состояние на диаграмме должно иметь входной переход, изображаемый потоком из стартового узла.

Применяются два способа построения STD. Первый способ заключается в идентификации всех возможных состояний и дальнейшем исследовании всех не бессмысленных связей (переходов) между ними. По второму способу сначала строится начальное состояние, затем следующие за ним и т.д.

В ситуации, когда число состояний и/или переходов велико, для проектирования спецификаций управления могут использоваться матрицы переходов состояний. В матрице переходов по вертикали указываются состояния, из которых осуществляется переход, а по горизонтали - состояния, в которые осуществляется переход. При этом каждый элемент матрицы содержит соответствующие условия и действия, обеспечивающие переход из "вертикального" состояния в "горизонтальное".

Информационные системы:

1. Пассажир бронирует и покупает билет на рейс
2. Клиент сдает автомобиль в автосервис
3. Покупатель оформляет кредит на покупку товара
4. Пассажир приходит на регистрацию рейса в аэропорт
5. Клиент снимает квартиру через агентство недвижимости
6. Определение списка студентов закрывших сессию в срок из указанной группы
7. Формирование заказа на изготовление мебели
8. Выдача книг в библиотеке
9. Заправка автомобилей
10. Формирование чека для оплаты покупок в супермаркете
11. Учет автомобилей на автостоянке и расчет прибыли
12. Формирование анкеты, проведение анкетирования и обработка результатов
13. Диспетчер задач на компьютере
14. Работа с группами пользователей, назначение прав доступа
15. Формирование классного журнала в школе
16. Печать фотографий и фотосувениров

ЛАБОРАТОРНАЯ РАБОТА № 12

Тема : Функциональные диаграммы.
Диаграммы потоков данных (DFD). Диаграммы сущность-связь (ERD).
Унифицированный язык моделирования UML. Диаграммы UML.

Цель: разработать функциональные диаграммы по предложенной информационной системе.

Исходные данные (задание):

1. Ознакомиться с теоретическим материалом по составлению функциональных диаграмм.
2. Рассмотреть предложенные функциональные диаграммы.
3. Сформировать функциональные диаграммы по предложенной информационной системе.
4. Оформить отчет по работе

Функциональными называют диаграммы, в первую очередь отражающие взаимосвязи функций разрабатываемого программного обеспечения.

Они создаются на ранних этапах проектирования систем, для того чтобы помочь проектировщику выявить основные функции и составные части проектируемой системы и, по возможности, обнаружить и устранить существенные ошибки. Современные методы структурного анализа и проектирования предоставляют разработчику определённые синтаксические и графические средства проектирования функциональных диаграмм информационных систем.

Методология SADT

Функциональная модель SADT отображает функциональную структуру объекта, т.е. производимые им действия и связи между этими действиями. Основные элементы этой методологии основываются на следующих концепциях:

· графическое представление блочного моделирования. На SADT-диаграмме функции представляется в виде блока, а интерфейсы входа/выхода в виде дуг, соответственно входящих в блок и выходящих из него. Интерфейсные дуги отображают взаимодействие функций друг с другом;

· строгость и точность отображения.

Правила SADT включают:

· уникальность меток и наименований;

· ограничение количества блоков на каждом уровне декомпозиции;

· синтаксические правила для графики;

· связность диаграмм;

· отделение организации от функции;

· разделение входов и управлений.

Диаграммы – главные компоненты модели, все функции программной системы и интерфейсы на них представлены как блоки и дуги. Место соединения дуги с блоком определяет тип интерфейса. Дуга, обозначающая управление, входит в блок сверху, в то время как информация, которая подвергается обработке, представляется дугой с левой стороны блока, а результаты обработки – это дуги с правой стороны. Механизм (человек или автоматизированная система), который осуществляет операцию, представляется в виде дуги, входящей в блок снизу (рис. 1).

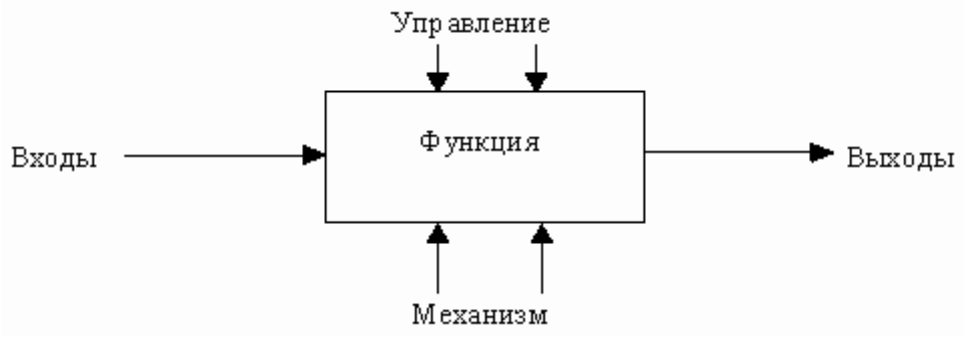


Рис. 1. Функциональный блок и интерфейсные дуги

Блоки на диаграмме размещают по «ступенчатой» схеме в соответствии с последовательностью их работы или *доминированием,* которое понимается как влияние, оказываемое одним блоком на другие. В функциональных диаграммах SADT различают пять типов влияний блоков друг на друга:

· вход-выход блока подается на вход блока с меньшим доминированием, т.е. следующего (рис. 2, *а);*

· управление – выход блока используется как управление для блока с меньшим доминированием (рис. 2, *б);*

· обратная связь по входу – выход блока подается на вход блока с большим доминированием (рис. 2, в);

· обратная связь по управлению – выход блока используется как управляющая информация для блока с большим доминированием (рис. 2, г);

· выход-исполнитель – выход блока используется как механизм для другого блока (рис. 2, д).

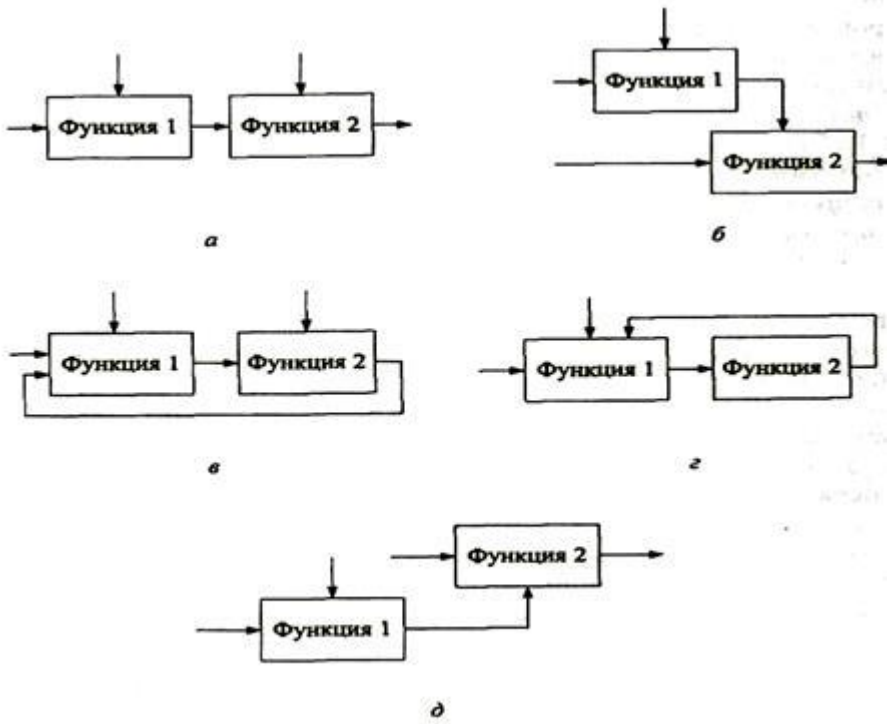


Рис. 2. Типы влияний блоков: а - вход; б - управление; в - обратная связь по входу; д - выход-исполнитель

.Иерархия диаграмм

Прежде всего, вся система представляется в виде простейшей компоненты – одного блока и дуг, представляющих собой интерфейсы с внешними по отношению к данной системе функциями. Имя блока является общим для всей системы.

Затем блок, который представляет систему в целом, детализируется на следующей диаграмме. Он представляется в виде нескольких блоков, соединенных интерфейсными

дугами (рис. 3). Каждый блок детальной диаграммы представляет собой подфункцию, границы которой определены интерфейсными дугами. Каждый из блоков детальной диаграммы может быть также детализирован на следующей в иерархии диаграмме. На каждом шаге декомпозиции более общая диаграмма называется родительской для более детальной диаграммы.

Все диаграммы связывают друг с другом иерархической нумерацией блоков: первый уровень – А0, второй – А1, А2 и т. п., третий – А11, А12, А13 и т. п., где первые цифры – номер родительского блока, а последняя – номер конкретного блока детальной диаграммы.

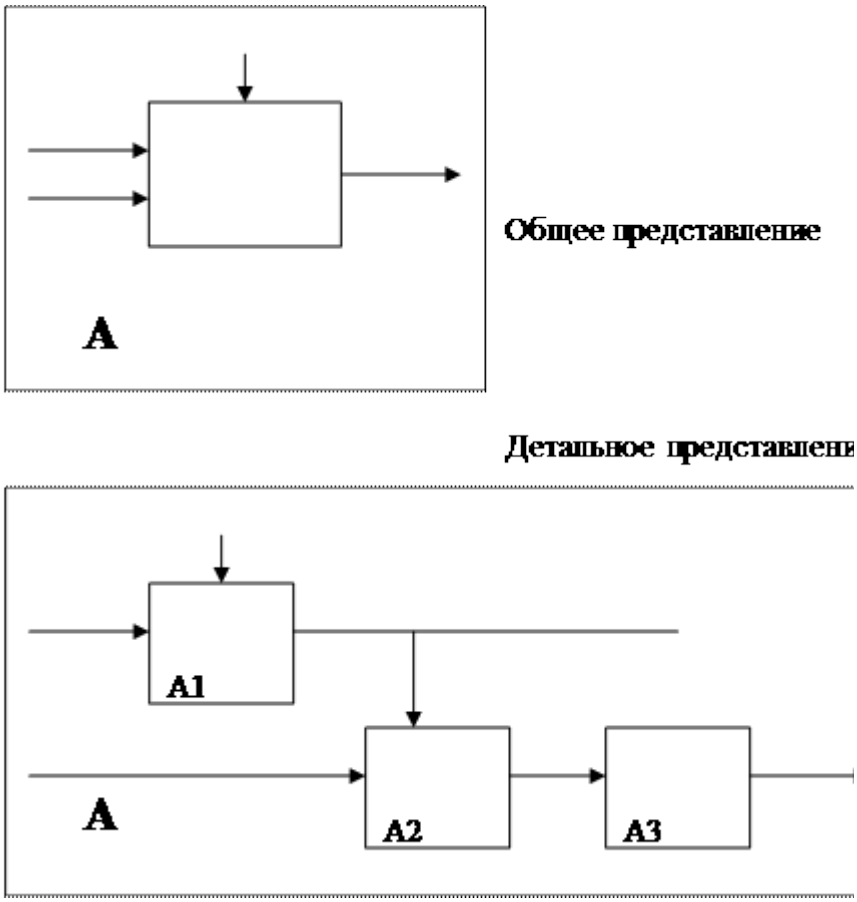


Рис. 3. Структура SADT-модели. Декомпозиция диаграмм

Дуги, входящие в блок и выходящие из него на диаграмме верхнего уровня, являются точно теми же самыми, что и дуги, входящие в диаграмму нижнего уровня и выходящие из нее, потому что блок и диаграмма представляют одну и ту же часть системы.

Пример 1.Разработку функциональных диаграмм продемонстрируем на примере уточнения спецификаций программы сортировки одномерного массива с использованием нескольких методов.

Диаграмма, представленная на рис. 4, *а,* является диаграммой верхнего уровня. Она иллюстрирует исходные данные программы и ожидаемые результаты.

Диаграмма, представленная на рис. 4, *б,* детализирует функции программы. На ней показаны три блока: **Меню**, **Сортировка**, **Вывод результата**. Для каждого блока определены исходные данные, управляющие воздействия и результаты. На детализирующей диаграмме используются следующие обозначения:

I1 – размер массива;

I2 – массив;

C1 – выбор метода;
R1 – вывод описания метода;
R2 – отсортированный массив.

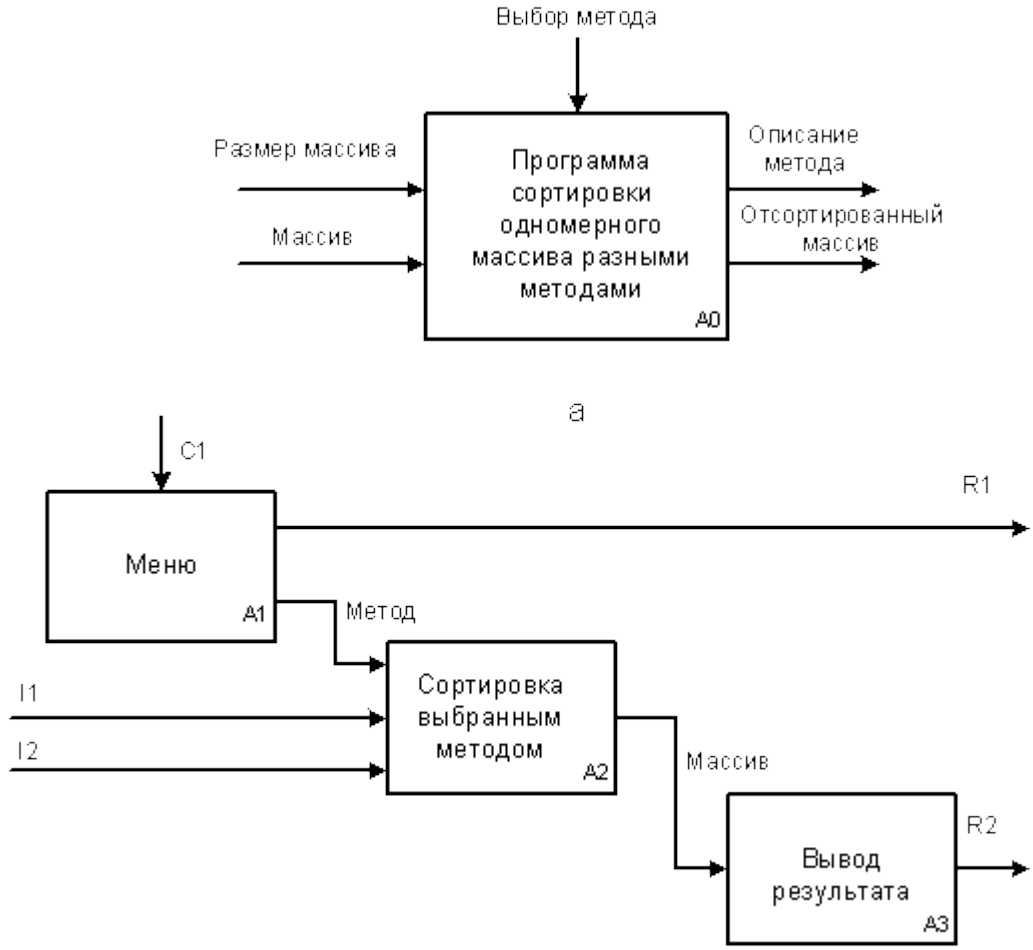


Рис. 4. Функциональные диаграммы для системы исследования функций:
а - диаграмма верхнего уровня; б - уточняющая диаграмма

Информационные системы:

1. Пассажир бронирует и покупает билет на рейс
2. Клиент сдает автомобиль в автосервис
3. Покупатель оформляет кредит на покупку товара
4. Пассажир приходит на регистрацию рейса в аэропорт
5. Клиент снимает квартиру через агентство недвижимости
6. Определение списка студентов закрывших сессию в срок из указанной группы
7. Формирование заказа на изготовление мебели
8. Выдача книг в библиотеке
9. Заправка автомобилей
10. Формирование чека для оплаты покупок в супермаркете
11. Учет автомобилей на автостоянке и расчет прибыли
12. Формирование анкеты, проведение анкетирования и обработка результатов
13. Диспетчер задач на компьютере
14. Работа с группами пользователей, назначение прав доступа
15. Формирование классного журнала в школе
16. Печать фотографий и фото сувениров

ЛАБОРАТОРНАЯ РАБОТА № 13

Тема: Диаграммы сущность – связь.

Цель: разработать диаграммы сущность-связь по предложенной информационной системе.

Базовыми понятиями ER-модели данных (ER – Entity-Relationship) являются: сущность, атрибут и связь.

Поскольку нотация Баркера является наиболее распространенной, в дальнейшем будем придерживаться именно ее.

Основные понятия ER-диаграмм

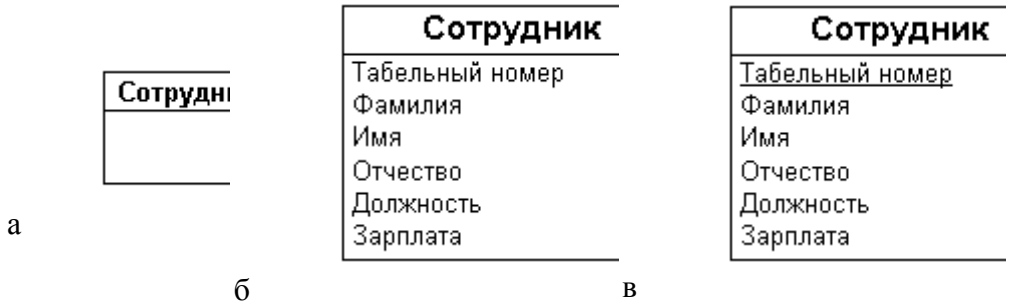
Сущность – это класс однотипных объектов, информация о которых должна быть учтена в модели. Сущность имеет наименование, выраженное существительным в единственном числе и обозначается в виде прямоугольника с наименованием (рис 1, а). Примерами сущностей могут быть такие классы объектов как «Студент», «Сотрудник», «Товар».

Экземпляр сущности– это конкретный представитель данной сущности. Например, конкретный представитель сущности «Студент» – «Максимов». Причем сущности должны иметь некоторые свойства, уникальные для каждого экземпляра этой сущности, для того чтобы различать экземпляры.

Атрибут сущности – это именованная характеристика, являющаяся некоторым свойством сущности. Наименование атрибута должно быть выражено существительным в единственном числе (возможно, с описательными оборотами или прилагательными). Примерами атрибутов сущности «Студент» могут быть такие атрибуты как «Номер зачетной книжки», «Фамилия», «Имя», «Пол», «Возраст», «Средний балл» и т.п. Атрибуты изображаются в прямоугольнике, обозначающем сущность (рис. 1, б).

Ключ сущности – это неизбыточный набор атрибутов, значения которых в совокупности являются уникальными для каждого экземпляра сущности. При удалении любого атрибута из ключа нарушается его уникальность. Ключей у сущности может быть несколько. На диаграмме ключевые атрибуты отображаются подчеркиванием (рис. 14, в).

Связь – это отношение одной сущности к другой или к самой себе. Возможно по одной сущности находить другие, связанные с ней. Например, связи между сущностями могут выражаться следующими фразами – «СОТРУДНИК может иметь несколько ДЕТЕЙ», «СОТРУДНИК обязан числиться точно в одном ОТДЕЛЕ». Графически связь изображается линией, соединяющей две сущности (рис. 2):



а б в

Рис. 1. Обозначения сущности в нотации Баркера:

а - без атрибутов; б -с указанием атрибутов; в -с ключевым атрибутом

Каждая связь имеет одно или два наименования. Наименование обычно выражается неопределенной формой глагола: «Продавать», «Быть проданным» и т.п. Каждое из наименований относится к своему концу связи. Иногда наименования не пишутся ввиду их очевидности.

Рис. 2. Пример связи между сущностями

Связь может иметь один из следующих **типов**:

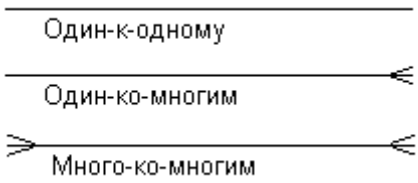


Рис. 3. Типы связей

Связь типа *один-к-одному* означает, что один экземпляр первой сущности связан точно с одним экземпляром второй сущности. Такая связь чаще всего свидетельствует о том, что мы неправильно разделили одну сущность, на две.

Связь типа *один-ко-многим* означает, что один экземпляр первой сущности связан с несколькими экземплярами второй сущности. Это наиболее часто используемый тип связи. Пример такой связи приведен на рис. 2.

Связь типа *много-ко-многим* означает, что каждый экземпляр первой сущности может быть связан с несколькими экземплярами второй сущности, и наоборот. Этот тип связи является временным, допустимым на ранних этапах разработки модели. В дальнейшем такую связь необходимо заменить двумя связями типа один-ко-многим путем создания промежуточной сущности.

Каждая связь может иметь одну из двух модальностей связи:

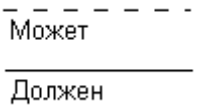
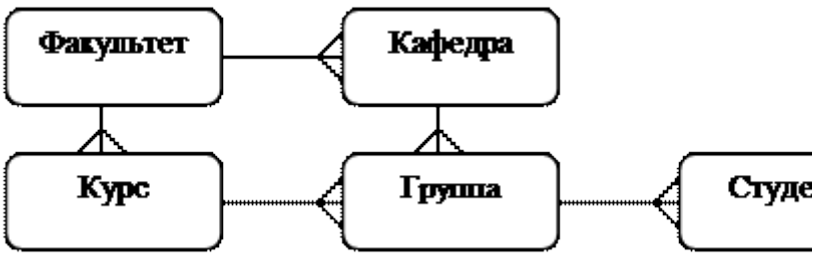


Рис. 4 Модальности связей

Связь может иметь разную модальность с разных концов как на рис. 4. Каждая связь может быть прочитана как слева направо, так и справа налево. Связь на рис. 2 читается так:

Слева направо: «Сотрудник может иметь несколько детей».

Справа налево: «Ребенок должен принадлежать точно одному сотруднику».



Пример разработки простой ER-диаграммы приведен на рис.5.

Рис. 5. Пример ER-диаграммы базы данных сведений о студентах

Информационные системы:

1. Пассажир бронирует и покупает билет на рейс
2. Клиент сдает автомобиль в автосервис
3. Покупатель оформляет кредит на покупку товара
4. Пассажир приходит на регистрацию рейса в аэропорт
5. Клиент снимает квартиру через агентство недвижимости
6. Определение списка студентов закрывших сессию в срок из указанной группы
7. Формирование заказа на изготовление мебели
8. Выдача книг в библиотеке
9. Заправка автомобилей
10. Формирование чека для оплаты покупок в супермаркете
11. Учет автомобилей на автостоянке и расчет прибыли
12. Формирование анкеты, проведение анкетирования и обработка результатов
13. Диспетчер задач на компьютере
14. Работа с группами пользователей, назначение прав доступа
15. Формирование классного журнала в школе
16. Печать фотографий и фотосувениров

Контрольные вопросы

1. Этапы разработки программного обеспечения.
2. Постановка задачи и предпроектные исследования.
3. Функциональные и эксплуатационные требования к программному продукту.
4. Составляющие эскизного проекта.
5. Спецификации и модели.
6. Диаграммы потоков данных.
7. Функциональные диаграммы.
8. Диаграммы переходов состояний.
9. Диаграммы «сущность-связь».
10. Факторы, влияющие на разработку программного обеспечения

ЛАБОРАТОРНАЯ РАБОТА №14

Тема: Унифицированный язык моделирования UML. Диаграммы UML.

Цель: Изучить унифицированный язык моделирования UML. Диаграммы UML.

UML – это унифицированный графический язык моделирования для описания, визуализации, проектирования и документирования ОО систем. UML призван поддерживать процесс моделирования ПС на основе ОО подхода, организовывать взаимосвязь концептуальных и программных понятий, отражать проблемы масштабирования сложных систем. Модели на UML используются на всех этапах жизненного цикла ПС, начиная с бизнес-анализа и заканчивая сопровождением системы. Разные организации могут применять UML по своему усмотрению в зависимости от своих проблемных областей и используемых технологий.К середине 90-х годов различными авторами было предложено несколько десятков методов ОО моделирования, каждый из которых использовал свою графическую нотацию. При этом любой их этих методов имел свои сильные стороны, но не позволял построить достаточно полную модель ПС, показать ее «со всех сторон», то есть, все необходимые проекции (См. статью 1). К тому же отсутствие стандарта ОО моделирования затрудняло для разработчиков выбор наиболее подходящего метода, что препятствовало широкому распространению ОО подхода к разработке ПС.

По запросу Object Management Group (OMG) – организации, ответственной за принятие стандартов в области объектных технологий и баз данных назревшая проблема унификации и стандартизации была решена авторами трех наиболее популярных ОО методов – Г.Бучем, Д.Рамбо и А.Джекобсоном, которые объединенными усилиями создали версию UML 1.1, утвержденную OMG в 1997 году в качестве стандарта.

UML – это язык

Любой язык состоит из словаря и правил комбинирования слов для получения осмысленных конструкций. Так, в частности, устроены языки программирования, таковым является и UML. Отличительной его особенностью является то, что словарь языка образуют графические элементы. Каждому графическому символу соответствует конкретная семантика, поэтому модель, созданная одним разработчиком, может однозначно быть понята другим, а также программным средством, интерпретирующим UML. Отсюда, в частности, следует, что модель ПС, представленная на UML, может автоматически быть переведена на ОО язык программирования (такой, как Java, C++, VisualBasic), то есть, при наличии хорошего инструментального средства визуального моделирования, поддерживающего UML, построив модель, мы получим и заготовку программного кода, соответствующего этой модели.

Следует подчеркнуть, что UML – это именно язык, а не метод. Он объясняет, из каких элементов создавать модели и как их читать, но ничего не говорит о том, какие модели и в каких случаях следует разрабатывать. Чтобы создать метод на базе UML, надо дополнить его описанием процесса разработки ПС. Примером такого процесса является Rational Unified Process, который будет рассматриваться в последующих статьях.

Словарь UML

Модель представляется в виде сущностей и отношений между ними, которые показываются на диаграммах.

Сущности – это абстракции, являющиеся основными элементами моделей. Имеется четыре типа сущностей – структурные (класс, интерфейс, компонент, вариант использования, кооперация, узел), поведенческие (взаимодействие, состояние), группирующие (пакеты) и аннотационные (комментарии). Каждый вид сущностей имеет свое графическое представление. Сущности будут подробно рассмотрены при изучении диаграмм.

Отношения показывают различные связи между сущностями. В UML определены следующие типы отношений:

- *Зависимость* показывает такую связь между двумя сущностями, когда изменение одной из них – независимой – может повлиять на семантику другой – зависимой. Зависимость изображается пунктирной стрелкой, направленной от зависимой сущности к независимой.
- *Ассоциация* – это структурное отношение, показывающее, что объекты одной сущности связаны с объектами другой. Графически ассоциация показывается в виде линии, соединяющей связываемые сущности. Ассоциации служат для осуществления навигации между объектами. Например, ассоциация между классами «Заказ» и «Товар» может быть использована для нахождения всех товаров, указанных в конкретном заказе – с одной стороны, или для нахождения всех заказов в которых есть данный товар, – с другой. Понятно, что в соответствующих программах должен быть реализован механизм, обеспечивающий такую навигацию. Если требуется навигация только в одном направлении, оно показывается стрелкой на конце ассоциации. Частным случаем ассоциации является агрегирование – отношение вида «целое» – «часть». Графически оно выделяется с помощью ромбика на конце около сущности-целого.
- *Обобщение* – это отношение между сущностью-родителем и сущностью-потомком. По существу, это отношение отражает свойство наследования для классов и объектов. Обобщение показывается в виде линии, заканчивающейся треугольничком направленным к родительской сущности. Потомок наследует структуру (атрибуты) и поведение (методы) родителя, но в то же время он может иметь новые элементы структуры и новые методы. UML допускает множественное наследование, когда сущность связана более чем с одной родительской сущностью.
- *Реализация* – отношение между сущностью, определяющей спецификацию поведения (интерфейс) с сущностью, определяющей реализацию этого поведения (класс, компонент). Это отношение обычно используется при моделировании компонент и будет подробнее описано в последующих статьях.

Диаграммы. В UML предусмотрены следующие диаграммы:

- Диаграммы, описывающие поведение системы:
    - Диаграммы состояний (State diagrams),
    - Диаграммы деятельностей (Activity diagrams),
    - Диаграммы объектов (Object diagrams),
    - Диаграммы последовательностей (Sequence diagrams),
    - Диаграммы взаимодействия (Collaboration diagrams);

- Диаграммы, описывающие физическую реализацию системы:
    - Диаграммы компонент (Component diagrams);
    - Диаграммы развертывания (Deployment diagrams).

Представление управления моделью. Пакеты.

Мы уже говорили о том, что для того чтобы модель была хорошо понимаемой человеком необходимо организовать ее иерархически, оставляя на каждом уровне иерархии небольшое число сущностей. UML включает средство организации иерархического представления модели – пакеты. Любая модель состоит из набора пакетов, которые могут содержать классы, варианты использования и прочие сущности и диаграммы. Пакет может включать другие пакеты, что позволяет создавать иерархии. В UML не предусмотрено отдельных диаграмм пакетов, но они могут присутствовать на других диаграммах. Пакет изображается в виде прямоугольника с закладкой.

Что обеспечивает UML.

- иерархическое описание сложной системы путем выделения пакетов;
- формализацию функциональных требований к системе с помощью аппарата вариантов использования;
- детализацию требований к системе путем построения диаграмм деятельностей и сценариев;
- выделение классов данных и построение концептуальной модели данных в виде диаграмм классов;
- выделение классов, описывающих пользовательский интерфейс, и создание схемы навигации экранов;
- описание процессов взаимодействия объектов при выполнении системных функций;
- описание поведения объектов в виде диаграмм деятельностей и состояний;
- описание программных компонент и их взаимодействия через интерфейсы;
- описание физической архитектуры системы.

И последнее…

Несмотря на всю привлекательность UML, его было бы затруднительно использовать при реальном моделировании ПС без инструментальных средств визуального моделирования. Такие средства позволяют оперативно представлять диаграммы на экране дисплея, документировать их, генерировать заготовки программных кодов на различных ОО языках программирования, создавать схемы баз данных. Большинство из них включают возможности реинжиниринга программных кодов – восстановления определенных проекций модели ПС путем автоматического анализа исходных кодов программ, что очень важно для обеспечения соответствия модели и кодов и при проектировании систем, наследующих функциональность систем-предшественников.

ЛАБОРАТОРНАЯ РАБОТА № 15

Тема: Инструментарий программирования. Драйверы и заглушки. Методы повышения информативности программ.

Цель: изучить элементы интерфейса Delphi.

Исходные данные (задание):

Запуск Delphi

Способов запустить среду существует множество (как и любой другой программы впрочем). Ярлык на рабочем столе, иконка на панели быстрого запуска, пункт в главном меню (Пуск - Программы - Borland Delphi n - Delphi n, где n - номер версии). Также есть удобный способ запустить Delphi через окно Пуск - Выполнить - ввести в этом окне delphi32. Более новые версии (2007, 2009, 2010) выпускаются уже не Borland, а

CodeGear, поэтому в главном меню группа называется CodeGear (Delphi входит в состав RAD Studio, поэтому может быть и название CodeGear RAD Studio). Из командной строки запуск осуществляется командой bds.

Delphi IDE

Вот и первое, возможно новое, для Вас слово. IDE (Integrated Development Environment) - интегрированная среда разработки программного обеспечения. После запуска Delphi перед Вами предстаёт эта самая среда. Состоит она из нескольких окон. Сейчас мы разберём, что это за окна и каково назначение каждого из них. В разных версиях Delphi эти окна могут выглядить немного по-разному, а некоторые и вообще могут отсутствовать. В данной статье будут приведены иллюстрации окон Delphi 7.

Итак, после запуска, наверное, Вы сразу обратите внимание, что среда в целом практически не отличается от других Windows-приложений. Все элементы стандартные. Главным окном можно считать то, которое содержит строку меню и панели инструментов. Вот строка меню:

Многие из этих пунктов стандартны. Если Вы установили русскую версию Delphi, то пункты будут называться примерно так: Файл, Правка, Поиск, Вид, Проект, Запуск, Компонент, База данных, Инструменты, Окно, Справка.

Как и во многих приложениях здесь есть панели инструментов. Они небольшие, кнопок на них немного, но всё самое основное как раз здесь и собрано. Панели инструментов выглядят примерно так:

Теперь рассмотрим те элементы, которых в обычных приложениях нет.

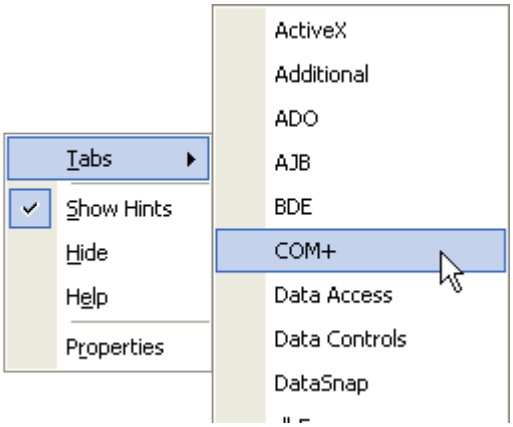
Палитра компонент (Component palette)

Палитра компонент - это набор вкладок, на каждой из которых расположены элементы. Именно с помощью этих элементов создаются интерфейсы программ. Все эти элементы принято называть *компонентами*. Среди компонент бывают как визуальные, так и невизуальные, но об этом мы поговорим позже. Вот как выглядит палитра компонент:

Её внешний вид практически одинаков во всех версиях Delphi. Да что там Delphi, такие же вкладки есть в любой среде объектно-ориентированного программирования (ООП), ибо это самый удобный способ предоставить выбор из сотен (а иногда даже тысяч) различных элементов.

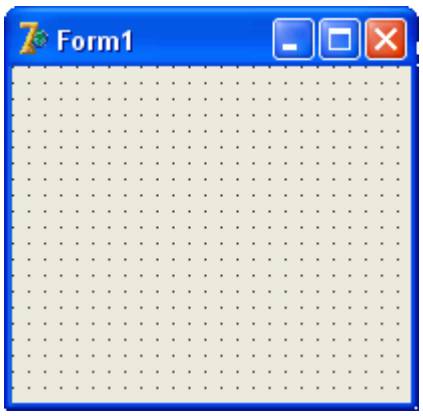
Переключение между вкладками осуществляется стандартным способом - щелчком по названию одной из вкладок. Сразу после установки в Delphi Вы можете видеть огромное количество вкладок. Они даже не помещаются на экране - для прокрутки созданы горизонтальные кнопки со стрелками. Также есть ещё один удобный способ перемещаться

по вкладкам - можно щёлкнуть правой кнопкой мыши по палитре компонент и в появившемся меню выбрать пункт Tabs - в результате откроется меню, где будут названия всех существующих вкладок в алфавитном порядке:



Дизайнер форм (Form Designer)

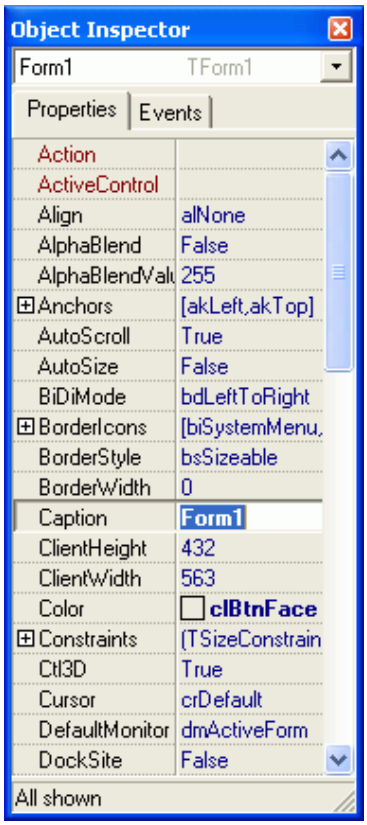
Это самое большое окно всей среды, которое изначально пустое. Именно это - заготовка окна Вашей программы. Здесь и будут размещаться все компоненты. Удобной составляющей дизайнера форм является сетка (множество точек). С помощью этой сетки компоненты удобно размещать на одном уровне, делать их одинаковых размеров и т.д. Это сделано для того, чтобы приложения соответствовали стандартам, установленным Microsoft. На этом мы ещё остановимся в одной из статей. Сетка является настраиваемой - можно изменить расстояние между точками, а можно её и вовсе отключить.



Инспектор объектов (Object Inspector)

Это окошко с двумя вкладками, каждая из которых состоит из двух колонок. В этом окне можно настроить параметры выбранного элемента и задействовать установленные события. Вкладки - *Properties* и *Events* (*Свойства* и *События* соответственно). Что это за свойства и что же такое события? По этому вопросу можно сказать очень много, это тема для отдельной статьи. А вкратце вот о чём речь. Допустим, у нас есть кнопка. Обыкновенная, какая используется в большинстве приложений. Примерами свойств этой кнопки могут быть её размеры (ширина, высота), текст, расположенный на ней и т.д. События - это предопределённые моменты реакции кнопки на какие-либо действия пользователя (либо действия со стороны операционной системы, внешних устройств и т.п.). Самый простой пример - щелчок по кнопке (так называемый "*клик*" - от слова *Click)*. Очевидно, что это событие произойдёт тогда, когда пользователь щёлкнет по кнопке, т.е. нажмёт её. У большинства компонент предусмотрены стандартные события. Как правило,

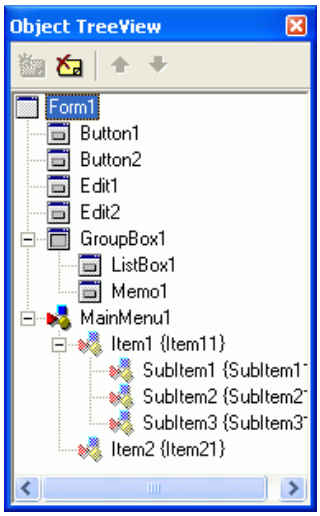
среди них есть все необходимые, которые могут понадобиться при создании программы. Однако можно создать и свои события как реакции на что-либо.



Дерево объектов (Object TreeView)

Это окошко появилось только в Delphi 6, в более ранних версиях его не было. В этом окне отображаются все элементы, какие есть на данной форме. Это сделано с целью упростить выбор компонентов для изменения их свойств в Object Inspector (далее - OI). Помимо того, что отображаются названия компонентов, рядом находятся маленькие графические значки, по которым можно определить, что это за объект. Элементы на форме не всегда автономны, т.е. самостоятельны, поэтому образуются иерархические связи - "деревья". Из-за этого окно и называется деревом объектов. В качестве простейшего примера иерархии объектов можно привести меню. Меню - это самостоятельный компонент, а вот его пункты - это "подчинённые" объекты. Пункт меню не может "висеть в воздухе" - он создан в конкретном меню.

Примечание: при динамическом создании пунктов меню они всё же могут просто находиться в памяти и не быть привязанными к какому-либо меню; данный пример приведён лишь для общего представления о связях между объектами.

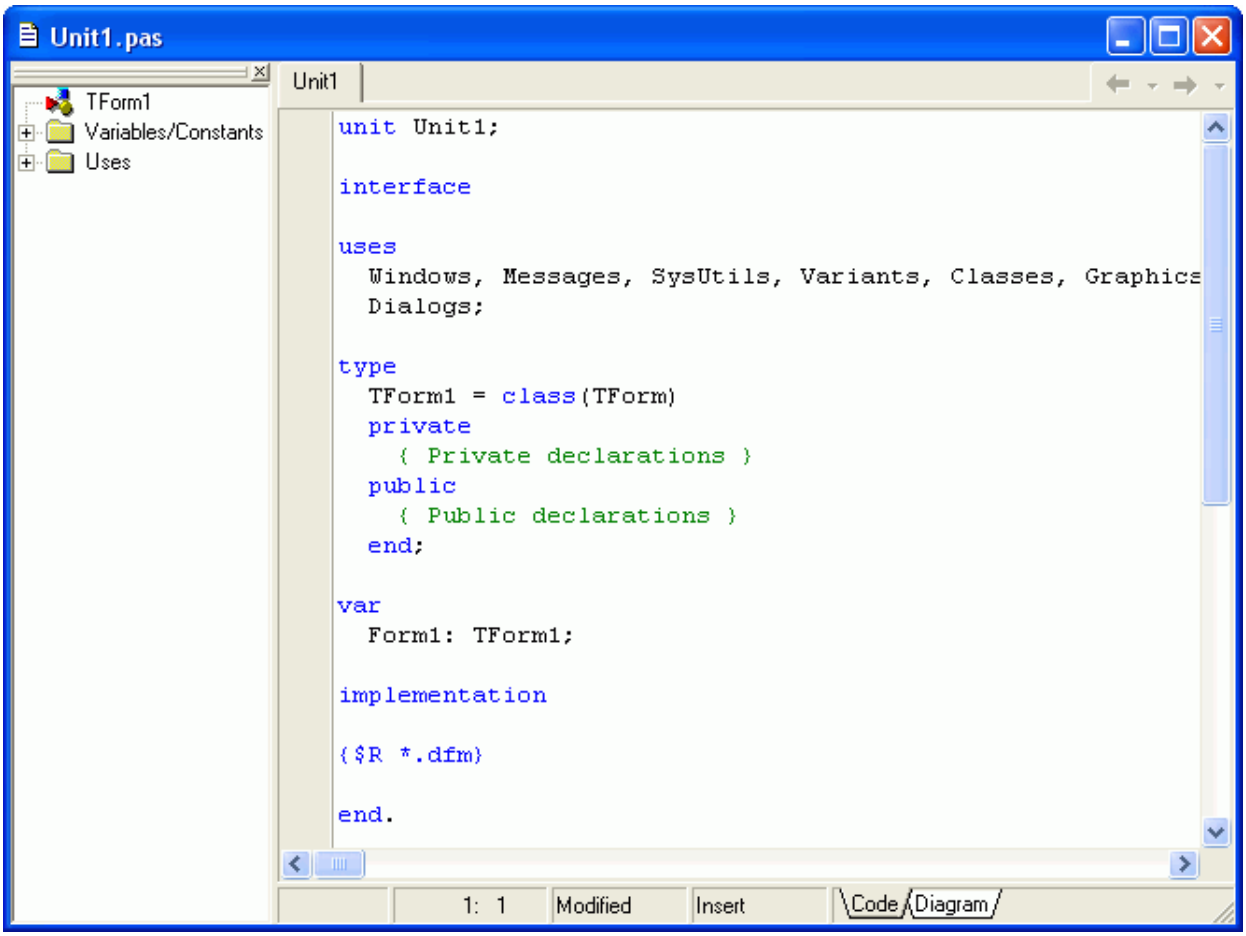


Редактор кода

Редактор кода представляет собой окно, похожее на обычный текстовый редактор за исключением некоторых дополнительных элементов. Основная область этого окна - поле редактирования. Именно здесь пишется текст программы. В отличие от языков программирования, работающих в текстовом режиме (Pascal, QBasic и т.п.) код программы здесь не пишется "с нуля". Как только Вы запустите Delphi и создадите новый проект, то, открыв редактор кода, увидите там уже часть написанной программы. Эти строки удалять ни в коем случае нельзя!

Окно редактора кода может содержать сразу несколько открытых файлов - переключение между ними осуществляется по закладкам вверху окна (на рисунке открыт только один файл - Unit1, поэтому закладка одна-единственная). В левой части окна расположено поле, называемое Code Explorer (*Обозреватель кода*). Здесь в виде дерева отображаются все типы, классы, свойства, методы, глобальные переменные и другие блоки, находящиеся в данном файле (модуле). Дело в том, что содержимое модуля состоит из отдельных участков. Назначение каждого из них мы рассмотрим чуть позже.

В нижней части окна расположена строка состояния, содержащая полезную информацию. В ней представлена текущая позиция курсора в тексте (номер строки : номер символа), текущий режим режима замены (Insert/Overwrite), информация о том, были ли внесены изменения в модуль с момента последнего сохранения и т.п.



Когда Вы попытаетесь запустить программу, то внизу появится информационное поле, в котором будут показаны сообщения об ошибках в тексте программы. Также в этом окне показываются полезные советы по оптимизации кода. В одной из статей мы разберём все эти сообщения более подробно. Если программа написана "идеально", т.е. ошибок нет и оптимизировать нечего, то окошко даже и не появится на экране.

Заключение

Итак, мы рассмотрели все основные элементы оболочки Delphi, которые используются в процессе работы. Конечно же, в Delphi существует множество других окон, но их назначение и способы вызова на экран мы будем рассматривать в процессе работы. Интерфейс в более новых версиях, чем Delphi 7, отличается, но все основные элементы те же самые, просто расположены они несколько иначе. При желании можно настроить оболочку по своему вкусу.

ЛАБОРАТОРНАЯ РАБОТА №16

Тема: Элементы интерфейса.

Цель: изучить элементы интерфейса Delphi.

Создайте новый проект. Поместите на форму объект TMemo, а затем TEdit так, чтобы он наполовину перекрывал TMemo, как показано на рис.13. Теперь выберите пункт меню Edit | Send to Back, что приведет к перемещению TEdit вглубь формы, за объект TMemo. Это называется изменением Z-порядка компонент. Буква Z используется потому, что

обычно математики обозначают третье измерение буквой Z. Так, X и Y используются для обозначения ширины и высоты, и Z используется для обозначения глубины.

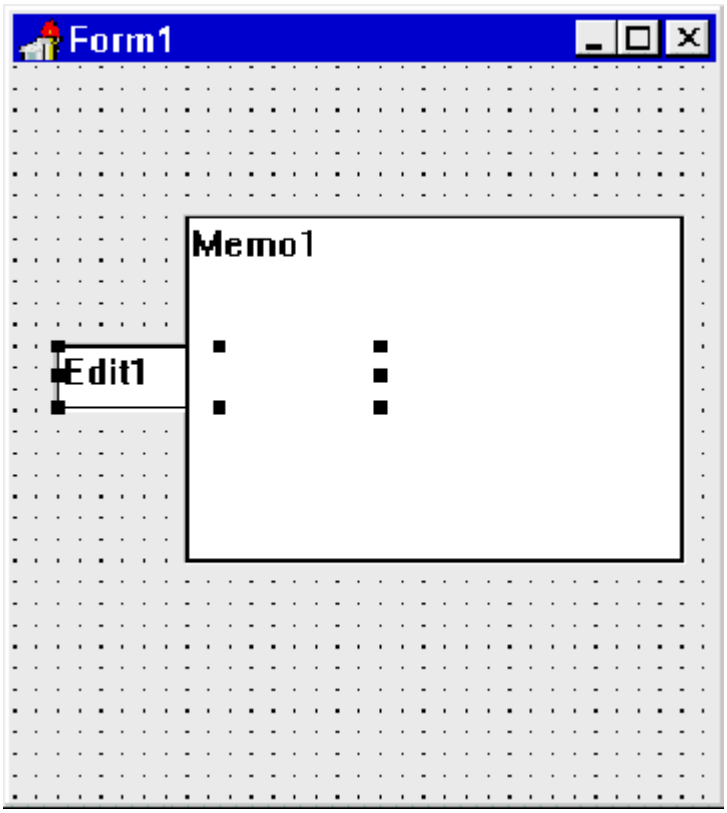


Рис.13: Объект TEdit перекрывается наполовину объектом TMemo.

Если Вы "потеряли" на форме какой-то объект, то найти его можно в списке Combobox'a, который находится в верхней части Инспектора Объектов.

Поместите кнопку TButton в нижнюю часть формы. Теперь растяните Инспектор Объектов так, чтобы свойства Name и Caption были видны одновременно на экране. Теперь измените имя кнопки на Terminate. Заметьте, что заголовок (Caption) поменялся в тот же момент. Такое двойное изменение наблюдается только если ранее не изменялось свойство Caption.

Текст, который Вы видите на поверхности кнопки - это содержимое свойства Caption, свойство Name служит для внутренних ссылок, Вы будете использовать его при написании кода программы. Если Вы откроете сейчас окно Редактора, то увидите следующий фрагмент кода:

```
    TForm1 = class(TForm)
      Edit1: TEdit;
Memo1: TMemo;
      Terminate: TButton;
      private
       { Private declarations }
      public
       { Public declarations }
    end;
```

В этом фрагменте кнопка TButton называется Terminate из-за того, что Вы присвоили это название свойству Name. Заметьте, что TMemo имеет имя, которое присваивается по умолчанию.

Перейдите на форму и дважды щелкните мышкой на объект TButton. Вы сразу попадете в окно Редактора, в котором увидите фрагмент кода вроде этого:

```
procedure TForm1.TerminateClick(Sender: TObject);
begin

end;
```

Данный код был создан автоматически и будет выполняться всякий раз, когда во время работы программы пользователь нажмет кнопку Terminate. Вдобавок, Вы можете видеть, что определение класса в начале файла теперь включает ссылку на метод TerminateClick:

```
  TForm1 = class(TForm)
    Edit1: TEdit;
    Memo1: TMemo;
    Terminate: TButton;
    procedure TerminateClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

Потратьте немного времени на усвоение последовательности действий, описанных выше. Изначально Вы смотрите на кнопку на форме. Вы делаете двойной щелчок на эту кнопку, и соответствующий фрагмент кода автоматически заносится в Редактор.

Теперь самое время написать строчку кода. Это очень простой код, состоящий из одного слова Close:

```
procedure TForm1.TerminateClick(Sender: TObject);
begin
  Close;
end;
```

Когда этот код исполняется, то главная форма (значит и все приложение) закрывается. Для проверки кода запустите программу и нажмите кнопку Terminate. Если все сделано правильно, программа закроется и Вы вернетесь в режим дизайна.

Прежде, чем перейти к следующему разделу, перейдите в Инспектор Объектов и измените значение свойства Name для кнопки на любое другое, например OK. Нажмите Enter для внесения изменений. Посмотрите в Редактор, Вы увидите, что код, написанный Вами изменился:

```
procedure TForm1.OkClick(Sender: TObject);
begin
  Close;
end;
```

Заметьте, что аналогичные изменения произошли и в определении класса:

```
TForm1 = class(TForm)
    Edit1: TEdit;
    Memo1: TMemo;
    Ok: TButton;
    procedure OkClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

Тьюторы (интерактивные обучающие программы)

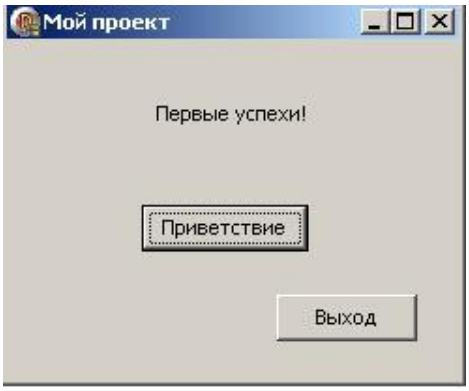
Delphi предоставляет тьютор, содержащий несколько тем и который можно запустить из пункта меню Help | Interactive Tutors. Тьютор запускается только если среда Delphi имеет все установки по умолчанию. Если конфигурация была изменена, то проще всего сохранить файл DELPHI.INI под другим именем и скопировать файл DELPHI.CBT в DELPHI.INI.

В первых двух темах дается краткий обзор Delphi и происходит обучение построению простейшего приложения.

Остальные темы посвящены построению пользовательского интерфейса: размещению объектов на форме, настройке их свойств и написанию обработчиков событий. А также созданию приложений, работающих с базами данных.

ЛАБОРАТОРНАЯ РАБОТА №17

Тема: Стиль программирования. Выбор языка программирования.
Цель: создать программу, выполняющую следующие действия.



После запуска программы по щелчку мышью на кнопке «**Приветствие»** появляется сообщение «**Первые успехи!**» (рис.1). Для выхода из программы необходимо щелкнуть мышью на кнопке «**Выход»**.

Рис.1

Новым в этой работе является:

- использование компонентов **Label** (метка) и **Button** (кнопка) палитры компонентов **Standard,**
- обработка события **OnClick** – нажатие кнопки.

План разработки программы

1. Откройте новый проект.
2. Разместите в форме экземпляры компонентов: метку **Label** и две кнопки **Button**

   (см. рис.2).

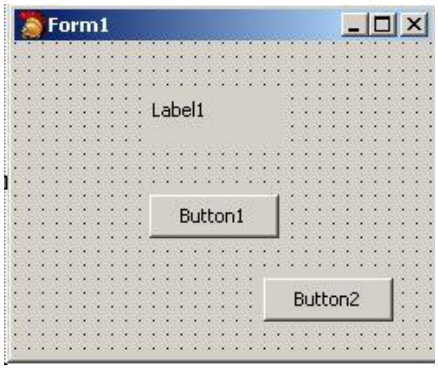


Рис.2

3. Выделите кнопку **Button2**, перейдите в **Object Inspector** на вкладку **Properties**, найдите свойство **Caption** (заголовок) и измените **Button2** на заголовок «**Выход**».

Перейдите на страницу **Events** окна **Object Inspector**, найдите событие **OnClick**, справа от него дважды щелкните мышью. Оказавшись в коде программы, точнее, в заготовке процедуры **TForm1.Button2Click**, напишите лишь одну команду **Close;** (обязательно поставьте точку с запятой после «Close»).

```
procedure TForm1.Button2Click(Sender: TObject); begin
   Close;
end;
```

---

4. Сохраните код программы и проект под именами, например, **Unit2.pas** и **Pr2.dpr**.
5. Запустите программу, затем закрыть окно проекта, щелкнув на кнопке «Выход».
6. Выделите форму, в свойстве **Caption** окна **Object Inspector** замените слово **Form1** на «Мой проект». Это и будет заголовком основного окна программы.
7. Выделите кнопку **Button1**, в свойстве **Caption** окна **Object Inspector** замените слово **Button1** на название копки «Приветствие». При необходимости увеличьте длину кнопки.
8. Не снимая выделения с кнопки **Button1**, перейдите на страницу **Events** окна **Object Inspector** и найдите событие **OnClick**, справа от него дважды щелкните мышью. Введите следующий код:

```
Label1.Caption:= ’Первые успехи!’;
```

9. Сохраните проект окончательно, запустите и протестируйте его.

Краткое описание плана разработки программы

В этом разделе показано, как можно кратко описать план разработки программы. Для краткости в дальнейшем будем использовать этот способ записи.

1. Откройте новый проект.
2. Разместите в форме экземпляры компонентов компоненты: метку **Label** и две кнопки **Button** (см. рис.5).

3. Выполните следующие действия:

Выделенный объект	Вкладка окна **Object Inspector**	Имя свойства/ Имя события	Значение/Действие
Button2	Properties	Caption	Выход
	Events	OnClick	Close;

4. Сохраните код программы и проект под именами, например, **Unit2.pas** и **Pr2.dpr**.
5. Запустите программу, затем закройте окно проекта кнопкой «Выход».
6. Выполните следующие действия:

Выделенный объект	Вкладка окна **Object Inspector**	Имя свойства/ Имя события	Значение/Действие
Form1	Properties	Caption	Мой проект
Button1	Properties	Caption	Приветствие
	Events	OnClick	Label1.Caption:= ’Первые успехи!’;

7. Сохраните проект, запустите и протестируйте его.

Задание для самостоятельного выполнения

1. Сделайте шрифт выводимой реплики «Первые успехи!» отличным от стандартного по виду, цвету и размеру.

Подсказка. В **Object Inspector** дважды щелкните на кнопку справа от названия свойства **Font** (шрифт), откроется окно выбора шрифта, его цвета и стиля.

2. Замените вид кнопки «Выход» на более привлекательный.

Подсказка. Для замены кнопки надо удалить существующую, а другую кнопку найдите в палитре компонентов на вкладке **Additional**. Она называется **BitBtn**. Затем измените ее вид с помощью свойства **Kind**.

3. Сделайте так, чтобы после нажатия кнопки «Приветствие» на экране появлялось сообщение «Первые и не последние!».

Подсказка. Измените значение свойства **Caption** метки **Label1** при реакции кнопки

Button1 на событие **OnClick**.

4. Запустите исполняемый файл Pr2.exe не в среде Delphi, а в Windows.

Подсказка. Выйдите из Delphi в Windows. Используйте диспетчер программ или проводник Windows.

Основная литература:

1. Долженко А.И. Технологии командной разработки программного обеспечения информационных систем [Электронный ресурс]/ Долженко А.И.— Электрон. текстовые данные.— М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016.— 300 c.— Режим доступа: http://www.iprbookshop.ru/39569.html.— ЭБС «IPRbooks»
2. Синицын С.В. Верификация программного обеспечения [Электронный ресурс] : учебное пособие / С.В. Синицын, Н.Ю. Налютин. — Электрон. текстовые данные. — Москва, Саратов: Интернет-Университет Информационных Технологий (ИНТУИТ), Вузовское образование, 2017. — 368 c. — 978-5-4487-0074-3. — Режим доступа: http://www.iprbookshop.ru/67396.html
3. Грекул В.И., Коровкина Н.Л., Куприянов Ю.В. Проектирование информационных систем. М.: Национальный открытый университет «ИНТУИТ», 2016, Режим доступа: http://www.iprbookshop.ru/67376.html

Дополнительная литература:

1. Рудаков А. Технология разработки программных продуктов: учебник. Изд.Academia. Среднее профессиональное образование. 2013 г. 208 стр.
2. Котляров В.П. Основы тестирования программного обеспечения [Электронный ресурс]/ Котляров В.П.— Электрон. текстовые данные.— М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016.— 334 c.— Режим доступа: http://www.iprbookshop.ru/62820.html.— ЭБС «IPRbooks»