Министерство образования и науки Российской Федерации
Федеральное агенство по образованию

Государственное образовательное учреждение
высшего профессионального образования
«Оренбургский государственный университет»

Кафедра системного анализа и управления

Ю.А. Ушаков, А.Л. Коннов

ТЕОРИЯ И ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ

Методические указания
к лабораторным работам. Часть 1

Рекомендовано к изданию Редакционно-издательским советом Государственного образовательного учреждения высшего профессионального образования «Оренбургский государственный университет»

Оренбург 2010

ББК 32.973-0181Я7
З 91
УДК 004.05(07)

Рецензент - кандидат технических наук, доцент Гаибова Т.В.

Ушаков Ю.А.
З 91 Теория и технология программирования: методические указания к лабораторным работам. Часть 1/ Ю.А. Ушаков, А.Л. Коннов; Оренбургский гос. ун-т. - Оренбург: ГОУ ОГУ, 2010. – 32 с.

Основное содержание: жизненный цикл ПО. Стиль программирования. Качество ПО. Модульное программирование. Отладка и тестирование. Документирование.

Методические указания предназначены для студентов, обучающихся по программам высшего профессионального образования по направлению 220100 Системный анализ и управление (бакалавриат), 230100 «Информатика и вычислительная техника» (бакалавриат), 511800 – «Математика, компьютерные науки» (бакалавриат) при изучении дисциплин «Теория и технология программирования», «Программирование на языке высокого уровня», «Программирование».

УДК 004.05(07)
ББК 32.973 – 0181я7

Содержание

Введение

Настоящие методические указания посвящены изложению принципов и методов анализа, синтеза и тестирования, используемых в инженерном цикле разработки сложных программных продуктов.

Первый раздел описывает жизненный цикл программного обеспечения (ПО). Описаны различные современные модели жизненных циклов как ПО в целом, так и отдельных аспектов (безопасности). Рассказывается о инструментах, помогающих в разработке приложений. Второй раздел посвящен лабораторной работе по тематике первого раздела.

Третий раздел описывает стили программирования как основу создания качественного ПО. Рассматриваются вопросы сопровождения, стандартизации. Четвертый раздел посвящен лабораторной работе по тематике третьего раздела. Пятый раздел описывает понятие качества программных средств. Рассмотрены вопросы надежности, понятности, функциональности. Шестой раздел посвящен лабораторной работе по тематике пятого раздела.

В седьмом разделе описывается особый подход к программированию - модульное программирование. Рассмотрены вопросы оптимизации модулей, структура и пример модульной программы. Восьмой раздел посвящен лабораторной работе по тематике седьмого раздела. Девятый раздел описывает аспекты отладки и тестирования ПО. Рассмотрены вопросы проектирования тестов, приведен пример, даны примеры отладки. Десятый раздел посвящен лабораторной работе по тематике девятого раздела.

Одиннадцатый раздел описывает документирование ПО. Рассматриваются вопросы пользовательской документации и документации сопровождения. Двенадцатый раздел посвящен лабораторной работе по тематике одинадцатого раздела.

Указания предназначены для студентов бакалаврского и магистерского уровней компьютерных специальностей.

1 Жизненный цикл ПО

В программных проектах, больших и малых, методология разработки программы используется для проектирования, разработки и сопровождения приложения. Эта методология может полностью отсутствовать при реализации малых проектов. Совершенно иначе выглядят проекты, в которых задействованы команды разработчиков и группы конечных пользователей, а сроки исполнения проектов исчисляются месяцами и годами совместной работы. В данном случае необходима строгая методология создания и реализации проектов, называемая Жизненным Циклом Разработки Программ (Software development life cycle) или ЖЦРП (SDLC).

SDLC может сильно отличаться от проекта к проекту и от руководителя проекта к руководителю проекта. Однако, обычно он состоит из следующих стадий (методология 1):

1) Анализ пожеланий и требований заказчика
2) Уточнение функциональных характеристик
3) Создание технического проекта (технического задания)
4) Реализация
5) Системное тестирование
6) Послереализационный обзол
7) Сопровождение

Быстрое макетирование (прототипирование) (rapid prototyping) - метод проектирования, разработки и изменения интерфейсов пользователя на лету. Используется для быстрого создания опытных образцов или работающей модели системы для демонстрации заказчику или проверки возможности реализации. Конечные пользователи должны тесно включаться в данный процесс, поскольку разработка интерфейса вместе с пользователем происходит значительно быстрее, нежели без него. Использование совместной разработки дает возможность подогнать интерфейс под пользователя за несколько коротких сессий. Термин используется в информационных технологиях для обозначения процесса быстрой разработки программного обеспечения (rapid application development , RAD).

Computer Aided Software Engineering (CASE) средства также играют огромную роль в сегодняшних инструментальных средствах разработки приложений. С мощными CASE-средствами процесс разработки приложений заметно упрощается. Проектировщик использует программные средства для создания и компоновки словарей данных, потоков данных и диаграм объекта, а в некоторых случаях прототипов процессов обработки данных и функционального кода.

Однако, использование CASE-средств разработки приложений не очень распространено в сфере разработки промышленных приложений. Это происходит по двум причинам. Во-первых, это ограниченность возможностей CASE-систем. Во-вторых, если CASE-система достаточно мощна и многофункциональна, то она требует больших временных затрат на ее освоение.

Для того, что бы не выбирать между различными формальными и инструментальными подходами, каждая команда разработчиков должна уметь

использовать общие инструменты разработки, а реализация всего жизненного цикла в целом ложится на менеджера проекта.

Следующие инструменты крайне удобна для разработки больших и средних проектов.

Техническое проектирование - это мост между функциональной спецификацией и фактической стадией кодирования. функциональная спецификация концентрирует внимание на требованиях и пожеланиях пользователя, а техническое проектирование должно ориентироваться на создание методов реализации данных требований. Только после того, как обе эти фазы завершены и акценты расставлены, программист может приступать к непосредственному кодированию.

Диаграмма зависимости объектов - уникальный способ для неинформированного программиста получить быстрый краткий обзор того, что система делает. Эта диаграмма показывает все объекты системы и связи между ними (один-к-одному, один-ко-многим и т.д.). Она может также показывать спецификации ключевых полей и другие связи по мере необходимости.

Структурная диаграмма - это высокоуровневое пректирование программных модулей и связи между ними, начиная с главного модуля и основных модулей, определенных ранее в функциональной спецификации (например, главное меню). Детализированная структурная диаграмма может также фключать информацию о передаваемых между модулями параметрах. Эта диаграмма - хороший способ отдельным членам команды программистов быстро выяснить общую структуру программы и решить все проблемы по интеграции отдельных модулей в систему. Это smоментальный снимоks разрабатываемой системы.

Детализация модулей - когда техническое проектирование высокого уровня закончено, выполняется в деталях проектирование низкого уровня. Для каждого модуля проектировщик должен определить все требования, включая передаваемые параметры, глобальные структуры и переменные, вызываемые подпрограммы и т.д. Эта информация важна для тех членов команды программистов, которые реализуют модули, взаимодействующие друг с другом. Хорошо спроектированная техническая спецификация снимает все проблемы, возникающие при соединении отдельных модулей программы в единое целое.

Библиотека функций - хотя библиотека функций внутреннего использования и не должна быть описана в проекционной документации, необходимо, чтобы хотя бы где-нибудь она была описана. Библиотека функций, с помощью которых реализуется проект - это очень важная часть любой разработки. Иногда отсутствие хорошего описания инструментальных средств, при помощи которых разрабатывается система, приводит к написанию программистом своих собственных инструментальных средств, дублирующих уже существующие.

Системные тесты - стадия тестирования системы начинается, когда все или большинство модулей системы уже завершены. Тестирование может состоять из трех отдельных фаз:

- Альфа тестирование (лабораторное).
- Бета тестирование (опытное).
- Приемочный тест.

Сопровождение программ - "ложка дегтя" для каждого программиста. Это всегда помеха при начале разработки какого-либо нового проекта, заставляющая отвлекаться от разработки проекта и возвращаться к старым программам и старым проблемам. Что делает эти шаги наиболее непривлекательными, так это плохо документированный код, недостаточно полное начальное проектирование и отсутствие внешней документации.

Компания Microsoft пропагандирует использование «Жизненного цикла безопасной разработки Microsoft Security Development Lifecycle (SDL)», показанный на рисунке 1.

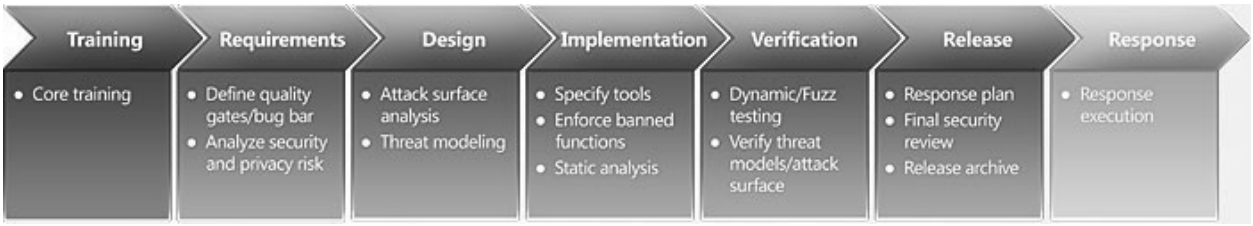

Рисунок 1 - Microsoft Security Development Lifecycle

Этапы данного подхода описаны ниже (методология 2):

Этап 1. Требования. На этапе разработки требований группа создания продукта может обдумать, как лучше всего встроить меры безопасности в процесс разработки; выявить основные задачи безопасности и повысить уровень безопасности ПО с самым незначительным влиянием на удобство заказчиков, планы и графики. Задачи:

- Анализ рисков конфиденциальности и безопасности
- Определение качественных порогов

Этап 2. Проектирование. На этапе проектирования определяются общие требования и структура ПО, устанавливаются лучшие методики проектирования. Задачи:

- Моделирование угроз
- Анализ уязвимостей

Этап 3. Реализация. На этапе реализации группа проекта устанавливает и использует лучшие методики разработки программного обеспечения. Задачи:

- Выбор инструментов
- Применение функций
- Статический анализ

Этап 4. Проверка. На этапе проверки программное обеспечение уже полностью функционально; оно проверяется на соответствие целям безопасности и конфиденциальности, заданным на этапах требований и проектирования. Задачи:

- Динамическое и случайное тестирование
- Проверка моделей угроз и уязвимостей

Этап 5. Выпуск. На этапе выпуска программный продукт готов к поступлению в продажу; идет разработка планов по последующему обслуживанию продукта. Задачи:

– План реагирования
– Итоговая проверка безопасности
– Архив выпусков

Этап 6. После SDL: реагирование.

2 Лабораторная работа 1

Необходимо воссоздать все этапы проектирование программного средства в соответствии с одной из методологий, представленных выше. Кроме того, для перехода между этапами, необходимо применять инструменты разработки, например техническое проектирование, детализацию модулей и т.д.

Лабораторная работа рассчитана на 3-5 человек. Один человек назначается менеджером проекта – он выполняет глобальное проектирование по этапам, распределяет этапы работ между остальными участниками.

Цель работы – научиться работать в команде над проектами, изучать, соблюдать и создавать спецификации для различных процессов, научиться декомпозировать задачи.

Задачи работы:

1) Определить требования - в формате интервью с заказчиком.
2) Формализовать требования.
3) Определить входную и выходную информацию.
4) Определить перечень входящих и исходящих документов.
5) Определить основные функции ПС.
6) Создать схему реализации проекта.
7) Декомпозировать каждый этап по функциональным признакам.
8) Определить входную и выходную информацию для каждого этапа.
9) Согласовать входную и выходную информацию с остальными разработчиками.
10) Специфицировать каждый этап документально.

Вариант 1

Разработать проект программного средства для учета и анализа данных о продажах.

Вариант 2

Разработать проект программного средства для организации документооборота в медицинском учреждении

Вариант 3

Разработать проект программного средства для учета и анализа данных о семьях, нуждающихся в улучшении жилищных условий

Вариант 4

Разработать проект программного средства для учета и анализа успеваемости студентов

Вариант 5

Разработать проект программного средства для организации телефонного справочника организации

Вариант 6

Разработать проект программного средства для учета и анализа данных об установленном ПО на компьютерах организации.

3 Стиль программирования

Стиль программирования связан с удобочитаемостью программы. Правила хорошего стиля программирования – это результат соглашения между опытными программистами.

Правило стандартизации стиля заключается в следующем: если существует более одного способа сделать что-либо и выбор произвольный, остановитесь на одном способе, и всегда его придерживайтесь. Программное средство представленное в хорошем стиле имеет комментарии (пояснительные, вводные иногда оглавления), значимые идентификаторы, хорошо воспринимаемый текст ПС.

Программисты GNU фонда Free Software Foundation (самого большого сообщества программистов) используют следующий стиль написания кода

Язык. Хорошим тоном является использование англоязычных идентификаторов, например вместо dengi использовать pay, вместо devushka – girl. Вопреки распространенному мнению, для того что бы использовать англоязычные идентификаторы, английского языка знать не нужно (хотя для программиста очень желательно), для этого достаточно лексикона из 400-500 наиболее употребимых слов.

Имена переменных и функций. Допустим у вас есть переменная или функция означающая "Убрать из строки все пробелы", программист, именующий идентификаторы в стиле операционных систем Unix записал бы ее название так "strsptr" или даже "strst". То есть все было бы максимально кратко и в нижнем регистре.

При таком подходе у исходных текстов программы имеется один большой недостаток. Если в тексте нет комментариев (а это скорее правило чем исключение), то через полгода просматривая исходник вы будете долго вспоминать, чего бы это значило - "if(!strst(mystr)) printf...".

Краткое название идентификатора уместно использовать только для именования индексных переменных в теле цикла или в качестве локальной переменной небольшой процедуры (таблица 1).

Таблица 1 – Пример стилей программирования

Стиль	С-подобные языки
1	2
Плохой	*for(int n= 0; n< nCount; n++)* *{* *printf("%d\n", n);* *}*
Хороший	*for(int nLineNo= 0; nLineNo< nLineCount; nLineNo++)* *{* *printf("%d\n", nLineNo);* *}*

Можно именовать идентификаторы и по другому. Допустим у вас есть все та же функция "Убрать из строки все пробелы", название может выглядеть примерно так "StrRemoveSpaces". В связи с таким подходом программа становится самодокументируемой и в этом случае можно уже действительно обойтись без комментариев. Хотя увлекаться самодокументированием тоже не следует, иначе если эта функция называется "RemoveAllSpacesFromThisString", то читать ее уже тоже затруднительно. Кстати, вот некоторое из WinAPI: FindClosePrinterChangeNotification, SystemTimeToTzSpecificLocalTime, INTERNET_ERROR_MASK_LOGIN_FAILURE_DISPLAY_ENTITY_BODY.

В программировании константы традиционно записываются в верхнем регистре, например COLO_BOT. Если вы хотите, чтобы другие программисты могли легко воспринимать ваш код, придерживайтесь этого правила.

Префиксы. Допустим у вас есть переменная имеющая тип DWORD, переменная в этом случае записывается например, так dwMyVar, то есть название переменной начинается с некоторых (или всех) букв взятых из названия ее типа. Так же иногда принято именовать глобальные переменные с префиксом g_... (g_dwMyVar). Переменные-члены класса в Microsoft Foundation Classes (MFC) начинаются с m_... (m_dwMyVar), а дальний указатель на строку которая заканчивается нуль-терминатором (LPSTR в WinAPI) так - lpsz... (lpszMyVar - Long Pointer String Zero).

Имеется великое множество таблиц префиксов переменных, каким следовать – решать разработчику. Главное, что бы в команде все разработчики придерживались одной таблицы.

О префиксах сейчас ходит много споров, насколько они нужны в современных языках. Многие программисты уже отказываются от этого так как компилятор сам проверяет типы данных.

Использование сокращений. Не стоит увлекаться использованием сокращений и аббревиатур в именовании идентификаторов, исключением могут быть только "общепринятые" сокращения, например: str -> string, src -> source, tbl -> table и т.д., так как если написать вместо server - srv, может или получиться путаница (возможно

это означает - service), или другой программист этого сокращения вообще не поймет.

Использование разделителей. Некоторые программисты используют в качестве разделителей слов в идентификаторах символ подчерка "_", имя нашей функции получилось бы таким - "str_remove_spaces". Некоторые вообще не используют разделителей ("strremovespaces" - читать затруднительно). Наиболее упорные используют такой стиль именования - "Str_Remove_Spaces". Остальные используют в качестве разделителей заглавные буквы - "strRemoveSpaces" или "StrRemoveSpaces".

Расстановка оператором начала и конца блока и отступы. Внутри любых управляющих конструкций операторы следует располагать с отступом на одинаковое число пробелов (таблица 2).

Таблица 2 – Пример расстановки начала и конца блока

C-подобные языки	Pascal-подобные языки
1	2
if(flag) *{* *cout << "Hello world!";* *}*	*if flag=1 then* *begin* *write('hello world');* *end;*

Пробелы вокруг символов. Бинарные операторы следует обрамлять пробелами:

```
// Неправильно
 $a=$b+$c*$d;
 // Правильно
 $a = $b + $c * $d;
```

Символ пробела ассоциируется с новым словом, поэтому формула читается не как непонятный набор символов, а как нечто осмысленное.

Комментарии. Расставляйте комментарии по принципу "чем больше, тем лучше" — пройдёт некоторое время и вы забудете, что делал тот или иной программный блок. Принято комментировать код на английском языке, так исторически сложилось. Или в крайнем случае транслитом. Но если программа пишется для себя, то комментарии модно писать на любом удобном языке. Примеры комментариев:

```
// Программный модуль index.php
echo "Hello world!";
/* Это многострочный комментарий в стиле C
   он охватывает несколько строк – не допускается
   вложенных комментариев
```

```
*/
echo "Hello world!";
```

К хорошему тону относится использование однострочных комментариев для короткого комментария, а многострочного — для комментария, охватывающего несколько строк. Не возбраняется использовать однострочные комментарии для большого текста, особенно в начале файла или важного блока кода

```
////////////////////////////////////////////////////////////////////////////////
// Гостевая книга
////////////////////////////////////////////////////////////////////////////////
```

При расстановке однострочных комментариев возможно два варианта: непосредственно перед выполняемым оператором

```
// Вывод текстовой строки в окно браузера
echo "Hello world!";
```

и после точки с запятой

```
echo "Hello world!"; // Вывод текстовой строки в окно браузера
```

Лучше придерживаться первого правила, так как строка получается длинной и плохо воспринимается читающим. Единственным оправданием использования такого комментария является комментирование закрывающейся скобки длинного программного блока, содержащего много вложенных блоков.

Переменные в функциях и составные переменные. Допустим, существует функция, которая имеет 10 переменных:

```
MyFunc(a,b,c,d,e,f,g,h,i,j);
```

При вызове такой функции очень сложно визуально определить где какой параметр располагается. Для отделения параметров используются новые строки, иногда сопровождаемые комментариями:

```
MyFunc(   '85',          //Weight
          'Ivan',        //Name
          'Ivanov',      //Family name
          1956,          //Birth year
          'M',           //Gender
          'Orenburg',    //City
          'Married',     // marital status
          2,             //Children
          33);           //money
```

4 Лабораторная работа 2

Необходимо создать программный код, в оформлении которого используется описанный стиль программирования.

Цель – выработать привычку к созданию удобочитаемого кода, привить навыки именования переменных, комментирования программы по ходу ее создания.

Задача– написать код, используя:

- Правила отступов
- Правила именования переменных
- Комментарии
- Пробелы
- Другие возможные стили

Вариант 1

Написать программу, реализующую ввод с клавиатуры пяти чисел, сложение первых трех и умножение следующих двух чисел, вывод на экран результатов этих операций с пояснениями

Вариант 2

Написать программу, , которая вычисляет площадь треугольника по формуле Герона (1). Стороны треугольника вводить с клавиатуры, промежуточные результаты выводить на экран.

$$S = \sqrt{p(p-a)(p-b)(p-c)} \qquad (1)$$

где

p — полупериметр треугольника: $p = (a+b+c)/2$,

a,b,c – стороны треугольника.

Вариант 3

Дана длина ребра куба. Написать программу для нахождения объема куба и площадь его поверхности

Вариант 4

Написать программу для нахождения периметра и площади прямоугольного треугольника по длинам двух катетов

Вариант 5

Дан диаметр окружности. Написать программу для нахождения ее длины и площади круга

Вариант 6

Написать программу для нахождения среднего арифметического и среднего геометрического двух положительных чисел

Вариант 7

Написать программу для нахождения длины сторон треугольника по заданным координатам его вершин

Вариант 8

Написать программу для нахождения A^{28}, используя шесть операций умножения;

Вариант 9

Написать программу которая распечатывает последнюю цифру заданного числа

Вариант 10

Написать программу для нахождения площади кольца, внутренний радиус которого равен *R1*, а внешний радиус равен *R2*

5 Качество программных средств. Надежность.

Качество кода может определяться различными критериями. Некоторые из них имеют значение только с точки зрения человека. Например, то, как отформатирован текст программы, совершенно не важно для компьютера, но может иметь серьёзное значение для последующего сопровождения. Многие из имеющихся стандартов оформления кода, определяющих специфичные для используемого языка соглашения и задающие ряд правил, улучшающих читаемость кода, имеют своей целью облегчить будущее сопровождение ПО, включающее отладку и обновление. Существуют и другие критерии, определяющие, «хорошо» ли написан код, например, такие, как структурированность — степень логического разбиения кода на ряд управляемых блоков.

- Читаемость кода
- Лёгкость поддержки, тестирования, отладки, исправления ошибок, изменения и портируемости
- Низкая сложность кода
- Низкое использование ресурсов: памяти и процессорного времени
- Корректная обработка исключительных ситуаций
- Низкое количество предупреждений при компиляции и линковке

Фактор качества ПО — это нефункциональное требование к программе, которое обычно не описывается в договоре с заказчиком, но, тем не менее, является желательным требованием, повышающим качество программы. Некоторые из факторов качества:

1) Понятность. Назначение ПО должно быть понятным, из самой программы и документации.

2) Полнота. Все необходимые части программы должны быть представлены и полностью реализованы.

3) Краткость. Отсутствие лишней, дублирующейся информации. Повторяющиеся части кода должны быть преобразованы в вызов общей процедуры. То же касается и документации.

4) Портируемость. Лёгкость в адаптации программы к другому окружению: другой архитектуре, платформе, операционной системе или её версии.

5) Согласованность. По всей программе и в документации должны использоваться одни и те же соглашения, форматы и обозначения.

6) Сопровождаемость. Насколько сложно изменить программу для удовлетворения новых требований. Это требование также указывает, что программа должна быть хорошо документирована, не слишком запутана, и иметь резерв роста по использованию ресурсов (память, процессор).

7) Тестируемость. Позволяет ли программа выполнить проверку приёмочных характеристик, поддерживается ли возможность измерения производительности.

8) Удобство использования. Простота и удобство использования программы. Это требование относится прежде всего к интерфейсу пользователя.

9) Надёжность. Отсутствие отказов и сбоев в работе программ, а также простота исправления дефектов и ошибок:

10) Структурированность.

11) Эффективность. Насколько рационально программа относится к ресурсам (память, процессор) при выполнении своих задач.

12) Безопасность.

Помимо технического взгляда на качество ПО, существует и оценка качества с позиции пользователя. Для этого аспекта качества иногда используют термин «usability». Довольно сложно получить эту оценку для заданного программного продукта. Наиболее важные из вопросов, влияющий на оценку:

- Является ли пользовательский интерфейс интуитивно понятным?
- Насколько просто выполнять простые, частые операции?
- Насколько легко выполняются сложные операции?
- Выдаёт ли программа понятные сообщения об ошибках?
- Всегда ли программа ведёт себя так как ожидается?
- Имеется ли документация и насколько она полна?
- Является ли интерфейс пользователя самоописательным/само-документирующим?
- Всегда ли задержки с ответом программы являются приемлемыми?

Самыми важными требованиями являются те, которые предъявляет заказчик. Например для программных средств, использующихся на производстве крайне важен параметр надежности. Для программных средств, которые планируется часто модернизировать – это сопровождаемость, согласованность и тестируемость.

Надежность ПО - есть свойство программного обеспечения своевременно выполнять в заранее указанных условиях эксплуатации вперед установленные функции. Надежность устанавливается по результатам работы ПО, т.е. при динамической проверке всех программ на множестве входной информации. Некорректное ПО заведомо ненадежно, однако и корректное ПО может быть ненадежным.

Существует огромное количество критериев надежности, типов и видов отказов ПО. Приведем здесь самые распространенные варианты обеспечения надежности.

Проверка формата и типа входных данных. Проверка входных данных осуществляется путем сравнения формата входного сообщения с заданным в спецификации, что не гарантирует от ошибок ввода данных, а только обеспечивает правильную форму записи.

Самой распространенной причиной сбоя ненадежного ПО является ввод строковой информации в поле, предназначенное для чисел. Например в программе есть код:

```
Var    A:integer;
……….
readln(a);
```

Что будет, если пользователь (случайно или нет) введет вместо числа букву? Программа аварийно завершит свою работу. То есть произойдет отказ.

Как этого избежать? Опытные программисты, при вводе данных всегда используют промежуточное хранилище в виде текста. Например строковую переменную. А преобразование в число производить с проверкой формата. Например функцией *Val(строка, результат, код_ошибки);*

```
Var a,cod:integer;s:string;
……..
readln(s);
val(s,a,cod);
if cod>0 then  {Ошибка}
        begin
  writeln('Ошибка ввода');
   …{реакция на ошибку – выход, повтор ввода….}
end;
```

Некоторые делают цикл ввода – ввод повторяется до тех пор, пока пользователь не введет правильное значение. Но в этом случае может возникнуть другой сбой – зацикливание. Всегда должна быть возможность альтернативного выхода из цикла.

```
Var s:string;a,cod:integer;
……..
```

```
Repeat {цикл}
readln(s);
val(s,a,cod);
if cod>0 then  {Ошибка}
        begin
  writeln('Ошибка ввода');
end;
Until (cod=0)       {выход если ввод правильный}
        or (s='');     {или есть пользователь просто нажал Enter –
                        альтернативный выход}
```

В визуальных средах разработки также имеется очень мощный инструмент – задание маски ввода. Этим можно сильно упростить ручной ввод.

Неоднозначная логика. Часто бывает такая ситуация – человек вводит число, нажимает «Далее», затем понимает, что допустил ошибку и нажимает «Назад». И попадает в совершенно другое место. Это отказ называется ошибкой логики. Для того, что бы таких ошибок избежать, необходимо заранее всю логику, особенно взаимодействия разных частей программы, описать и графически представить.

Ошибки вычислений. Очень распространенная причина отказа. Например происходит деление на 0. Или переполнение типа. Тут сложно дать общую рекомендацию. Необходимо в каждом случае выполнять разные проверки. Например при делении на какую-нибудь переменную:

```
If B<>0 then
  C:=A/B
Else
 begin
  Write('Деление на 0, произошла ошибка');
   Exit;
 end;
```

В этом случае программа сообщит об ошибке и не будет выполнять неправильных действий.

Проверка границ. Часто бывает случай, когда необходимо, что бы параметр был в каком либо диапазоне. Например больше 1. Необходимо вводить проверку на диапазон, например:

```
a:=random(10)/5;
if a>1 then
b:=log(a);
       else
begin
 write ('Ошибка. A<=1');
  exit;
end;
```

Автоматическая реакция на ошибки. Многие среды программирования содержат инструменты для автоматической обработки ошибок. Например в Delphi есть такая конструкция

Try
Тут код программы
Except
Message('Ошибка');
End;

Если реакция на ошибку не важна, а важно корректное завершение процедуры или программы (запись данных в файл, вывод сведений) то существует другая конструкция

Try
Тут код программы
Finally
Message('Поздравляю, все прошло');
End;

В этом случае пользователь не увидит сообщения об ошибке, а программа корректно продолжит работу. Блок Finally будет выполнен в любом случае, была ошибка или нет.

6 Лабораторная работа 3

Необходимо написать надежную программу, которой не страшен некорректный ввод данных с клавиатуры.

Цель – освоить приемы повышения надежности ПО.

Задачи – Модернизировать лабораторную работу 2 в соответствии с требованиями надежности.

7 Модульное программирование

Модульное программирование является развитием и совершенствованием процедурного программирования и библиотек специальных программ. Основная черта модульного программирования - стандартизация интерфейса между отдельными программными единицами.

Модуль - это отдельная функционально-законченная программная единица, которая структурно оформляется стандартным образом по отношению к компилятору и по отношению к объединению ее с другими аналогичными единицами и загрузке. Как правило, каждый модуль содержит паспорт, в котором указаны все основные его характеристики: язык программирования, объем, входные и выходные переменные, их формат, ограничения на них, точки входа, параметры настройки и т.д.

Программы разбиваются на модули для того, чтобы:

- упростить их разработку и реализацию;
- облегчить чтение программ;
- упростить их настройку и модификацию;
- облегчить работу с данными, имеющими сложную структуру;
- избежать чрезмерной детализации алгоритмов;
- обеспечить более выгодное размещение программ в памяти ЭВМ.

Модульное программирование - это искусство разбиения задачи на некоторое число различных модулей, умение широко использовать стандартные модули путем их параметрической настройки, автоматизация сборки готовых модулей из библиотек, банков модулей.

Основные концепции модульного программирования:

- каждый модуль реализует единственную независимую функцию;
- каждый модуль имеет единственную точку входа и выхода;
- размер модуля по возможности должен быть минимизирован;
- каждый модуль может быть разработан и закодирован различными членами бригады программистов и может быть отдельно протестирован;
- вся система построена из модулей;
- модуль не должен давать побочных эффектов;
- каждый модуль не зависит от того, как реализованы другие модули.

При таком подходе сложная система разделяется на несколько частей, одновременно создаваемых различными программистами. Каждый модуль реализует единственную функцию. Размер модуля невелик, поэтому тестирование управляемо и может быть проведено тщательным образом. После кодирования и тестирования всех модулей происходит их интеграция, и тестируется вся система.

При сопровождении тестируется и отлаживается только тот модуль, который плохо работает. Очевидны преимущества в облегчении написания и тестирования программ, уменьшается стоимость их сопровождения.

Простой пример модульного программного средства «Калькулятор» представлен ниже (рисунок 2):

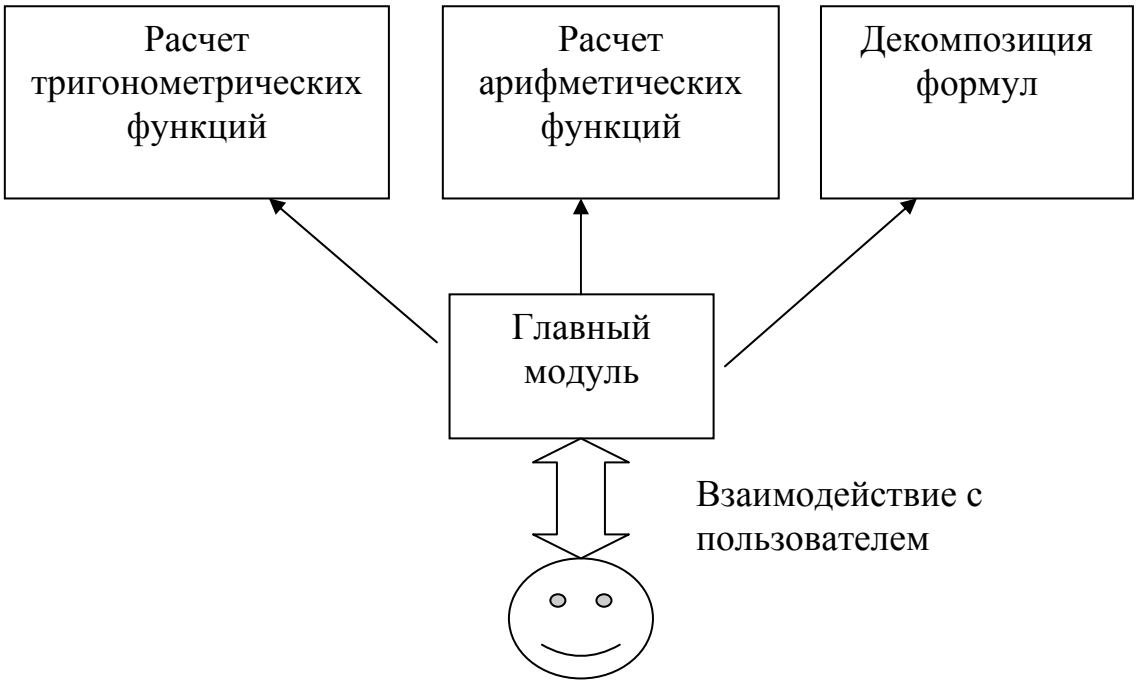

Рисунок 2 – Схема модульной программы – калькулятора

Каждый модуль реализует только одну функцию. Главный модуль осуществляет вызов требуемого модуля, передачу данных от пользователя и возвращает результаты выполнения пользователю.

Реализация модулей в большинстве языков одинаковая – подключение файла с модулем. Только в некоторых языках модули должны оформляться особым образом. Например (таблица 3):

Таблица 3 – Объявление модулей

Тип	С-подобный язык	Pascal-подобный язык
1	2	3
Объявление библиотеки (модуля)	*Объявления доступных структур, переменных, функций…* *…….* *КОД*	*Unit module1;* *Interface* *Объявление доступных структур, переменных, функций…* *Implementation* *КОД* *End.*
Вызов модуля	#include “module1.h”;	Uses module1;

8 Лабораторная работа 4

Работа выполняется группой 3-5 человек. Выбирается менеджер проекта – он будет делать главный модуль. Остальные модернизируют задания лабораторной работы 2 до состояния модуля с функциями. Каждый модуль должен содержать описание своих входных и выходных данных, типов и структур. По этим данным будет создан центральный модуль.

Цель – Научиться создавать модульные программы, трудиться в команде над реальным проектом.

Задачи

1) Модернизировать задачи до состояния модуля.
2) Составить спецификацию модуля
3) Создать главный модуль, который будет вызывать остальные модули в соответствии с их спецификацией.

9 Отладка и тестирование

Производство современного ПО происходит на фоне высоких требований, предъявляемых к качеству создаваемых программ и значительной сложности выполняемых ими функций. Для обеспечения надежности программных продуктов приходится выявлять ошибки на всех этапах проектирования, начиная с этапа анализа требований и заканчивая устранением ошибок на стадии сопровождения. И если на этапе анализа требований стремятся исключить логические ошибки той деятельности, под которую пишется ПО, то на остальных этапах от целого ряда ошибок стремятся избавится комплексом мер, начиная от декомпозиций и заканчивая “Идеальной” технологией программирования - технология, которая по некоторому достаточно неформальному описанию объекта программирования автоматически генерирует текст синтаксически и семантически корректной программы.

Компании-производители пытаются повысить качество программных продуктов с помощью тестирования. Существуют тестовые методики, использующие специальные драйверы, автоматически проверяющие промежуточные версии разрабатываемого продукта на различном аппаратном обеспечении. Другой способ проверки - бета-тестирование. В этом случае разработчики программного обеспечения разрешают пользователям попробовать предварительные версии продуктов.

Тестирование (testing), - процесс выполнения программы (или части программы) с намерением (или целью) найти ошибки.

Контроль (verification) — попытка найти ошибки, выполняя программу в тестовой, или моделируемой, среде.

Испытание (validation) — попытка найти ошибки, выполняя программу в заданной реальной среде.

Отладка (debugging) не является разновидностью тестирования. Хотя слова «отладка» и «тестирование» часто используются как синонимы, под ними подразумеваются разные виды деятельности. Отладка направлена на установление точной природы известной ошибки, а затем — на исправление этой ошибки. Эти два вида деятельности связаны — результаты тестирования являются исходными данными для отладки.

Тестирование модуля, или автономное тестирование (module testing, unit testing) — контроль отдельного программного модуля, обычно в изолированной среде (т. е. изолированно от всех остальных модулей). Тестирование модуля иногда включает также математическое доказательство. Иногда осложнено закрытостью модуля.

Тестирование программного обеспечения охватывает целый ряд видов деятельности, аналогичных последовательности процессов разработки программного обеспечения. В него входят:

1) постановка задачи для теста;
2) проектирование теста;
3) написание тестов;
4) тестирование тестов;
5) выполнение тестов;
6) изучение результатов тестирования.

Решающую роль играет проектирование тестов. Возможен целый ряд подходов к стратегии проектирования тестов. Чтобы ориентироваться в них, рассмотрим два крайних подхода. Первый состоит в том, что тесты проектируются на основе внешних спецификаций программ и модулей, либо спецификаций сопряжения программы или модуля. Программа при этом рассматривается как черный ящик (стратегия ‘черного ящика’). Существо такого подхода – проверить соответствует ли программа внешним спецификациям. При этом логика модуля совершенно не принимается во внимание.

Второй подход основан на анализе логики программы (стратегия ‘белого ящика’). Существо подхода – в проверке каждого пути, каждой ветви алгоритма. При этом внешняя спецификация во внимание не принимается.

Например, существует программа, на вход которой должны подаваться 2 числа, на выходе – из сумма.

1 ПЗ – необходимо проверить надежность и корректность программы для всех возможных входных данных;

2 тест должен содержать как корректные данные, так и некорректные, например целые числа, дробные числа, числа разделенные запятой, нечисловые значения

3 Тест представлен в таблице 4

Таблица 4 – Тест для проверки программы

Номер теста	Вход 1	Вход 2	Выход
1	2	3	4
1	1	1	2
2	1.1	1	2.1

Продолжение таблицы 4

1	2	3	4
3	1	1.1	2.1
4	Один	1	Ошибка
5	1	Один	Ошибка
6	Один	Один	Ошибка
7	1,1	1	Ошибка
8	1	1,1	Ошибка
9	1,1	1,1	Ошибка
10	100000000000000	100000000000000	200000000000000
11	1E+300	1E+300	2e+300
12	1e+500	1e+300	Ошибка

4 Тесты даются разработчику, может он тоже что нибудь добавит

5 Прогон тестов. Результаты представлены в таблице 5

Результат ошибка в колонку «Выход» ставиться только в случае корректного сообщения об ошибке. Если программа выдала некорректный результат, то в столбец «Корректность» ставиться «-», если программа выдала системную ошибку, зависла и т.п., то «-» ставиться в колонку «Надежность».

Таблица 5 – Результат тестирования

Номер теста	Выход	Корректность	Надежность
1	2	3	4
1	4		
1	2		
2	2.1		
3	2.1		
4	152		
5	Ошибка		
6	-	-	-
7	Ошибка		
8	Ошибка		
9	-	-	
10	200000000000000		
11	2e+300		
12	-	-	-

6 По результатам тестирования видно, что программа в ряде случаем некорректно работает, а при определенных условиях вызывает сбой. Для устранения недостатков необходима отладка.

Отладка. Наиболее распространенными и наименее эффективными для отладки являются так называемые методы ‘грубой силы’. К ним относят:

- отладку с использованием дампа памяти;
- отладку с использованием операторов печати по всей программе;

- отладку с использованием автоматических средств (среды разработки).

При программировании в среде разработки, чаще всего используется встроенный отладчик. В других случаях чаще используется печать переменных. Например, при тестировании было замечен аномальное значение выходного параметра. Что бы узнать причину, необходимо проследить путь вычисления переменной

```
a=10;b=1;
while (b<100)
{
        a=a*b/2;
  print “A+”+a+”; B=”+b+”\n”;
  b++;
}
```

Все значения на протяжении цикла будут напечатаны, по их величинам можно отследить момент возникновения сбоя.

10 Лабораторная работа 5

Необходимо полностью выполнить все пункты тестирования для программы из лабораторной работы 4.

Цель – Научиться проектировать и проводить тесты ПС, интерпретировать результаты, выполнять отладку.

Задачи –

1) постановка задачи для теста;
2) проектирование теста;
3) написание тестов;
4) тестирование тестов;
5) выполнение тестов;
6) изучение результатов тестирования
7) отладка ПС

11 Документирование

При разработке документации необходимо придерживаться требований стандартов ЕСПД.

Основу отечественной нормативной базы в области документирования ПС составляет комплекс стандартов Единой системы программной документации (ЕСПД). Основная и большая часть комплекса ЕСПД была разработана в 70-е и 80-е годы. Сейчас этот комплекс представляет собой систему межгосударственных стандартов стран СНГ (ГОСТ), действующих на территории Российской Федерации на основе межгосударственного соглашения по стандартизации.

Стандарты ЕСПД в основном охватывают ту часть документации, которая создается в процессе разработки ПС, и связаны, по большей части, с документированием функциональных характеристик ПС. Следует отметить, что стандарты ЕСПД (ГОСТ 19) носят рекомендательный характер. Впрочем, это относится и ко всем другим стандартам в области ПС (ГОСТ 34, Международному стандарту ISO/IEC, и др.). Дело в том, что в соответствии с Законом РФ "О стандартизации" эти стандарты становятся обязательными на контрактной основе - то есть при ссылке на них в договоре на разработку (поставку) ПС.

ГОСТ 19.105-78 ЕСПД. Общие требования к программным документам. Настоящий стандарт устанавливает общие требования к оформлению программных документов для вычислительных машин, комплексов и систем, независимо от их назначения и области применения и предусмотренных стандартами Единой системы программной документации (ЕСПД) для любого способа выполнения документов на различных носителях данных.

Программный документ может быть представлен на различных типах носителей данных и состоит из следующих условных частей:

- титульной;
- информационной;
- основной.

Правила оформления документа и его частей на каждом носителе данных устанавливаются стандартами ЕСПД на правила оформления документов на соответствующих носителях данных.

ГОСТ 19.106-78 ЕСПД. Требования к программным документам, выполненным печатным способом. Программные документы оформляют:

1) на листах формата А4 (ГОСТ 2.301-68) при изготовлении документа машинописным или рукописным способом;

2) допускается оформление на листах формата А3;

3) при машинном способе выполнения документа допускаются отклонения размеров листов, соответствующих форматам А4 и А3, определяемые возможностями применяемых технических средств; на листах форматов А4 и А3, предусматриваемых выходными характеристиками устройств вывода данных, при изготовлении документа машинным способом;

4) на листах типографических форматов при изготовлении документа типографским способом.

Расположение материалов программного документа осуществляется в следующей последовательности:

1) титульная часть:
2) лист утверждения (не входит в общее количество листов документа);
3) титульный лист (первый лист документа);
4) информационная часть:
5) аннотация;
6) лист содержания;
7) основная часть:
8) текст документа (с рисунками, таблицами и т.п.)
9) перечень терминов и их определений;
10) перечень сокращений;
11) приложения;
12) предметный указатель;
13) перечень ссылочных документов;
14) часть регистрации изменений:
15) лист регистрации изменений.

Перечень терминов и их определений, перечень сокращений, приложения, предметных указатель, перечень ссылочных документов выполняются при необходимости.

Следующий стандарт ориентирован на документирование результирующего продукта разработки.

ГОСТ 19.402-78 ЕСПД. Описание программы. Состав документа "Описание программы" в своей содержательной части может дополняться разделами и пунктами, почерпнутыми из стандартов для других описательных документов и руководств: ГОСТ 19.404-79 ЕСПД. Пояснительная записка, ГОСТ 19.502-78 ЕСПД. Описание применения, ГОСТ 19.503-79 ЕСПД. Руководство системного программиста, ГОСТ 19.504-79 ЕСПД. Руководство программиста, ГОСТ 19.505-79 ЕСПД. Руководство оператора.

Есть также группа стандартов, определяющая требования к фиксации всего набора программ и ПД, которые оформляются для передачи ПС. Они порождают лаконичные документы учетного характера и могут быть полезны для упорядочения всего хозяйства программ и ПД (ведь очень часто требуется просто навести элементарный порядок!). Есть и стандарты, определяющие правила ведения документов в "хозяйстве" ПС.

ГОСТ 19.301-79 ЕСПД. Программа и методика испытаний. Он (в адаптированном виде) может использоваться для разработки документов планирования и проведения испытательных работ по оценке готовности и качества ПС.

ГОСТ 19.701-90 ЕСПД. Схемы алгоритмов, программ, данных и систем. Обозначения условные графические и правила выполнения. Он устанавливает правила выполнения схем, используемых для отображения различных видов задач

обработки данных и средств их решения и полностью соответствует стандарту ИСО 5807:1985.

Наряду с ЕСПД на межгосударственном уровне действуют еще два стандарта, также относящихся к документированию ПС и принятых не так давно, как большая часть ГОСТ ЕСПД.

ГОСТ 19781-90 Обеспечение систем обработки информации программное. Термины и определения. Разработан взамен ГОСТ 19.781-83 и ГОСТ 19.004-80 и устанавливает термины и определения понятий в области программного обеспечения (ПО) систем обработки данных (СОД), применяемые во всех видах документации и литературы, входящих в сферу работ по стандартизации или использующих результаты этих работ.

ГОСТ 28388-89 Системы обработки информации. Документы на магнитных носителях данных. Порядок выполнения и обращения. Распространяется не только на программные, но и на конструкторские, технологические и другие проектные документы, выполняемые на магнитных носителях.

Документы управления разработкой ПС. Нужны что бы ПС могло сопровождаться и модернизироваться. Практически все ГОСТы описывают именно этот вид документации, поэтому не будем заострять на ней внимание.

Документы, входящие в состав ПС. Собственно, это и есть те документы, которые будет видеть пользователь. Они описывают программы ПС как с точки зрения их применения пользователями, так и с точки зрения их разработчиков и сопроводителей (в соответствии с назначением ПС). Здесь следует отметить, что эти документы будут использоваться не только на стадии эксплуатации ПС (в ее фазах применения и сопровождения), но и на стадии разработки для управления процессом разработки (вместе с рабочими документами) – во всяком случае они должны быть проверены (протестированы) на соответствие программам ПС. Эти документы образуют два комплекта с разным назначением:

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

Пользовательская документация ПС (user documentation) объясняет пользователям, как они должны действовать, чтобы применить данное ПС. Она необходима, если ПС предполагает какое-либо взаимодействие с пользователями. К такой документации относятся документы, которыми руководствуется пользователь при инсталяции ПС (при установке ПС с соответствующей настройкой на среду применения ПС), при применении ПС для решения своих задач и при управлении ПС (например, когда данное ПС взаимодействует с другими системами). Эти документы частично затрагивают вопросы сопровождения ПС, но не касаются вопросов, связанных с модификацией программ.

Документация по сопровождению ПС (system documentation) описывает ПС с точки зрения ее разработки. Эта документация необходима, если ПС предполагает изучение того, как оно устроена (сконструирована), и модернизацию его программ. Как уже отмечалось, сопровождение – это продолжающаяся разработка. Поэтому в случае необходимости модернизации ПС к этой работе привлекается специальная команда разработчиков-сопроводителей. Этой команде придется иметь дело с такой же документацией, которая определяла деятельность команды первоначальных

(основных) разработчиков ПС, – с той лишь разницей, что эта документация для команды разработчиков-сопроводителей будет, как правило, чужой (она создавалась другой командой). Команда разработчиков-сопроводителей должна будет изучать эту документацию, чтобы понять строение и процесс разработки модернизируемого ПС, и внести в эту документацию необходимые изменения, повторяя в значительной степени технологические процессы, с помощью которых создавалось первоначальное ПС.

12 Лабораторная работа 6

Необходимо создать пакет документации для сопровождения ПС, а также пользовательскую документацию к лабораторной работе 5.

Цель – научиться документировать ПС с точки зрения разработчика и с точки зрения программиста.

Задачи – необходимо создать следующие документы:

1) Руководство по инсталляции
2) Руководство по эксплуатации
3) Внешнее описание ПС
4) Описание архитектуры ПС, включая внешнюю спецификацию каждой ее программы
5) Для каждой программы ПС – описание ее модульной структуры, включая внешнюю спецификацию каждого включенного в нее модуля.
6) Для каждого модуля – его спецификация и описание его строения .
7) Тексты модулей на выбранном языке программирования.

Список использованных источников

1 ANSI/IEEE Std 1008-1987, IEEE Standard for Software Unit Testing.

2 ANSI/IEEE Std 1012-1986, IEEE Standard for Software Verification and Validation Plans.

3 ANSI/IEEE Std 1016-1987, IEEE Recommended Practice for Software Design Description.

4 ANSI/IEEE Std 1063-1988, IEEE Standard for Software User Documentation.

5 ANSI/IEEE Std 829-1983, IEEE Standard for Software Test Documentation.

6 ANSI/IEEE Std 830-1984, IEEE Guide for Software Requirements Specification.

7 ANSI/IEEE Std 983-1986, IEEE Guide for Software Quality Assurance Planning.

8 Абрамов С.А. Элементы программирования. - М.: Наука, 1982.-С. 85-94.

9 Агафонов В.Н. Спецификация программ: понятийные средства и их организация. - Новосибирск: Наука (Сибирское отделение), 1987.- 350 с.

10 Борисов В.М. Разработка пакетов программ вычислительного типа. – М.: Издательство МГУ, 1990. – 123 с.

11 Боэм Б., Дж. Браун Дж., Каспар Х. и др. Характеристики качества программного обеспечения. - М.: Мир, 1981. – 200 с.

12 Боэм Б.У. Инженерное проектирование программного обеспечения.- М.:Радио и связь, 1985.- 512 с.

13 Ван Тассел Д. Стиль, разработка, эффективность, отладка и испытание программ. - М.: Мир, 1985. - С. 179-295.

14 Вендеров А.М. Проектирование программного обеспечения экономических информационных систем. – М.: Финансы и статистика, 2002.- 348 с.

15 Жоголев Е.А. Введение в технологию программирования: Конспект лекций. - М.: "ДИАЛОГ-МГУ", 1994. – 150 с.

16 Жоголев Е.А. Технологические основы модульного программирования // Программирование.- 1980.- №2. - С. 44-49.

17 Зелковец М, Шоу А., Гэннон Дж. Принципы разработки программного обеспечения. - М.: Мир, 1982. - С. 11.

18 Зиглер К. Методы проектирования программных систем. – М.: Мир, 1985 .- 328 с.

19 Шураков В.В. Надежность программного обеспечения систем обработки данных. – М.: Статистика, 1981.- 216 с.