ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ОРЕНБУРГСКИЙ ГОСУДАРСТВЕННЫЙ АГРАРНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ДЛЯ ОБУЧАЮЩИХСЯ ПО ОСВОЕНИЮ ДИСЦИПЛИНЫ

Б1.В.07 Технология разработки программного обеспечения

Направление подготовки (специальность)
09.04.01 Информатика и вычислительная техника

Профиль образовательной программы
“Автоматизированные системы обработки информации и управления**”**

Форма обучения очная

СОДЕРЖАНИЕ

1. Тематическое содержание дисциплины

1.1. Тема 1: «Особенности разработки программного обеспечения»(33 часа).

1.1.1. Перечень и краткое содержание рассматриваемых вопросов:

1. Специфика разработки.

Прежде всего, следует отметить некоторое противостояние: неформальный характер требований к ПС (постановки задачи) и понятия ошибки в нем, но формализованный основной объект разработки - программы ПС. Тем самым разработка ПС содержит определенные этапы формализации, а переход от неформального к формальному существенно неформален.

Разработка ПС носит творческий характер (на каждом шаге приходится делать какой - либо выбор, принимать какое - либо решение), а не сводится к выполнению какой - либо последовательности регламентированных действий. Тем самым эта разработка ближе к процессу проектирования каких - либо сложных устройств, но никак не к их массовому производству. Этот творческий характер разработки ПС сохраняется до самого ее конца.

Следует отметить также особенность продукта разработки. Он представляет собой некоторую совокупность текстов (т.е. статических объектов), смысл же (семантика) этих текстов выражается процессами обработки данных и действиями пользователей, запускающих эти процессы (т.е. является динамическим). Это предопределяет выбор разработчиком ряда специфичных приемов, методов и средств.

Продукт разработки имеет и другую специфическую особенность: ПС при своем использовании (эксплуатации) не расходуется и не расходует используемых ресурсов.

2. Порядок разработки.

При разработке программного модуля целесообразно придерживаться следующего порядка:

- изучение и проверка спецификации модуля, выбор языка программирования;
- выбор алгоритма и структуры данных;
- программирование (кодирование) модуля;
- шлифовка текста модуля;
- проверка модуля;
- компиляция модуля.

Первый шаг разработки программного модуля в значительной степени представляет собой смежный контроль структуры программы снизу: изучая спецификацию модуля, разработчик должен убедиться, что она ему понятна и достаточна для разработки этого модуля. В завершении этого шага выбирается язык программирования: хотя язык программирования может быть уже предопределен для всего ПС, все же в ряде случаев (если система программирования это допускает) может быть выбран другой язык, более подходящий для реализации данного модуля (например, язык ассемблера).

На втором шаге разработки программного модуля необходимо выяснить, не известны ли уже какие-либо алгоритмы для решения поставленной и или близкой к ней задачи. И если найдется подходящий алгоритм, то целесообразно им воспользоваться. Выбор подходящих структур данных, которые будут использоваться при выполнении модулем своих функций, в значительной степени предопределяет логику и качественные показатели разрабатываемого модуля, поэтому его следует рассматривать как весьма ответственное решение.

На третьем шаге осуществляется построение текста модуля на выбранном языке

программирования. Обилие всевозможных деталей, которые должны быть учтены при реализации функций, указанных в спецификации модуля, легко могут привести к созданию весьма запутанного текста, содержащего массу ошибок и неточностей. Искать ошибки в таком модуле и вносить в него требуемые изменения может оказаться весьма трудоемкой задачей. Поэтому весьма важно для построения текста модуля пользоваться технологически обоснованной и практически проверенной дисциплиной программирования. Впервые на это обратил внимание Дейкстра, сформулировав и обосновав основные принципы структурного программирования. На этих принципах базируются многие дисциплины программирования, широко применяемые на практике. Наиболее распространенной является дисциплина пошаговой детализации.

Следующий шаг разработки модуля связан с приведением текста модуля к завершенному виду в соответствии со спецификацией качества ПС. При программировании модуля разработчик основное внимание уделяет правильности реализации функций модуля, оставляя недоработанными комментарии и допуская некоторые нарушения требований к стилю программы. При шлифовке текста модуля он должен отредактировать имеющиеся в тексте комментарии и, возможно, включить в него дополнительные комментарии с целью обеспечить требуемые примитивы качества. С этой же целью производится редактирование текста программы для выполнения стилистических требований.

3. Статистика ошибок и дефектов в комплексах программ.

Статистика ошибок и дефектов в комплексах программ и их характеристики в конкретных типах проектов ПС могут служить ориентирами для разработчиков при распределении ресурсов в жизненном цикле ПС и предохранять их от излишнего оптимизма при оценке достигнутого качества программных продуктов. Источниками ошибок в ПС являются специалисты — конкретные люди с их индивидуальными особенностями, квалификацией, талантом и опытом. При этом можно выделить предсказуемые модификации, расширения и совершенствования ПС и изменения, обусловленные выявлением случайных, непредсказуемых дефектов и ошибок. Вследствие этого плотность потоков и размеры необходимых корректировок в модулях и компонентах при разработке и сопровождении ПС могут различаться в десяток раз. Однако в крупных комплексах программ статистика и распределение типов выполняемых изменений для коллективов разных специалистов нивелируются и проявляются достаточно общие закономерности, которые могут использоваться как ориентиры при их выявлении и систематизации. Этому могут помогать оценки типовых дефектов, модификаций и корректировок путем их накопления и обобщения по опыту создания определенных классов ПС в конкретных предприятиях.

К понятию «риски» относятся негативные события и их величины, отражающие потери, убытки или ущерб от процессов или продуктов, вызванные дефектамипри проектировании требований, недостатками обоснования проектов ПС, а также при последующих этапах разработки, реализации и всего жизненного цикла комплексов программ. В ЖЦ ПС не всегда удается достигнуть требуемого положительного эффекта и может проявляться некоторый ущерб — риск в создаваемых проектах, программных продуктах и их характеристиках. Риски проявляются, как негативные последствия дефектов функционирования и применения ПС, которые способны нанести ущерб системе, внешней среде или пользователю в результате отклонения характеристик объектов или процессов от заданных требованиями заказчика, согласованными с разработчиками.

1.2. Тема 2: «Архитектура программного средства» (35 часов).

1.2.1. Перечень и краткое содержание рассматриваемых вопросов:

1. 1. Виды архитектур.

Основные задачи разработки архитектуры ПС:

- Выделение программных подсистем и отображение на них внешних функций (заданных во внешнем описании) ПС;
- определение способов взаимодействия между выделенными программными подсистемами.

С учетом принимаемых на этом этапе решений производится дальнейшая конкретизация и функциональных спецификаций.

Различают следующие основные классы архитектур программных средств:

- цельная программа;
- комплекс автономно выполняемых программ;
- слоистая программная система;
- коллектив параллельно выполняемых программ.

Цельная программа представляет вырожденный случай архитектуры ПС: в состав ПС входит только одна программа. Такую архитектуру выбирают обычно в том случае, когда ПС должно выполнять одну какую-либо ярко выраженную функцию и ее реализация не представляется слишком сложной. Естественно, что такая архитектура не требует какого-либо описания (кроме фиксации класса архитектуры), так как отображение внешних функций на эту программу тривиально, а определять способ взаимодействия не требуется (в силу отсутствия какого-либо внешнего взаимодействия программы, кроме как взаимодействия ее с пользователем, а последнее описывается в документации по применению ПС).

Комплекс автономно выполняемых программ состоит из набора программ, такого, что:

- любая из этих программ может быть активизирована (запущена) пользователем;
- при выполнении активизированной программы другие программы этого набора не могут быть активизированы до тех пор, пока не закончит выполнение активизированная программа;
- все программы этого набора применятся к одной и той же информационной среде.

Таким образом, программы этого набора по управлению не взаимодействуют - взаимодействие между ними осуществляется только через общую информационную среду.

Слоистая программная система состоит из некоторой упорядоченной совокупности программных подсистем, называемых слоями, такой, что:

- на каждом слое ничего не известно о свойствах (и даже существовании) последующих (более высоких) слоев;
- каждый слой может взаимодействовать по управлению (обращаться к компонентам) с непосредственно предшествующим (более низким) слоем через заранее определенный интерфейс, ничего не зная о внутреннем строении всех предшествующих слоев;
- каждый слой располагает определенными ресурсами, которые он либо скрывает от других слоев, либо предоставляет непосредственно последующему слою (через указанный интерфейс) некоторые их абстракции.

Таким образом, в слоистой программной системе каждый слой может реализовать некоторую абстракцию данных. Связи между слоями ограничены передачей значений параметров обращения каждого слоя к смежному снизу слою и выдачей результатов этого

обращения от нижнего слоя верхнему. Недопустимо использование глобальных данных несколькими слоями.

2. Архитектурные функции.

Для обеспечения взаимодействия между подсистемами в ряде случаев не требуется создавать какие-либо дополнительные программные компоненты (помимо реализации внешних функций) - для этого может быть достаточно заранее фиксированных соглашений и стандартных возможностей базового программного обеспечения (операционной системы). Так в комплексе автономно выполняемых программ для обеспечения взаимодействия достаточно описания (спецификации) общей внешней информационной среды и возможностей операционной системы для запуска программ. В слоистой программной системе может оказаться достаточным спецификации выделенных программных слоев и обычный аппарат обращения к процедурам. В программном конвейере взаимодействие между программами также может обеспечивать операционная система (как это имеет место в операционной системе UNIX).

Однако в ряде случаев для обеспечения взаимодействия между программными подсистемами может потребоваться создание дополнительных программных компонент. Так для управления работой комплекса автономно выполняемых программ часто создают специализированный командный интерпретатор, более удобный (в данной предметной области) для подготовки требуемой внешней информационной среды и запуска требуемой программы, чем базовый командный интерпретатор используемой операционной системы. В слоистых программных системах может быть создан особый аппарат обращения к процедурам слоя (например, обеспечивающий параллельное выполнение этих процедур). В коллективе параллельно действующих программ для управления портами сообщений требуется специальная программная подсистема. Такие программные компоненты реализуют не внешние функции ПС, а функции, возникшие в результате разработки архитектуры этого ПС. В связи с этим такие функции мы будем называть архитектурными.

3. Контроль архитектуры программных средств.

Для контроля архитектуры ПС используется смежный контроль и ручная имитация.

Смежный контроль архитектуры ПС сверху - это ее контроль разработчиками внешнего описания: разработчиками спецификации качества и разработчиками функциональной спецификации. Смежный контроль архитектуры ПС снизу - это ее контроль потенциальными разработчиками программных подсистем, входящих в состав ПС в соответствии с разработанной архитектурой.

Ручная имитация архитектуры ПС производится аналогично ручной имитации функциональной спецификации, только целью этого контроля является проверка взаимодействия между программными подсистемами. Так же как и в случае ручной имитации функциональной спецификации ПС должны быть сначала подготовлены тесты. Затем группа разработчиков должна для каждого такого теста имитировать работу каждой программной подсистемы, входящей в состав ПС. При этом работу каждой подсистемы имитирует один какой-либо разработчик (не автор архитектуры), тщательно выполняя все взаимодействия этой подсистемы с другими подсистемами (точнее, с разработчиками, их имитирующими) в соответствии с разработанной архитектурой ПС. Тем самым обеспечивается имитационное функционирование ПС в целом в рамках проверяемой архитектуры.

1.3. Тема 3: «Обеспечение функциональности и надежности программного средства» (36 часов).

1.3.1. Перечень и краткое содержание рассматриваемых вопросов:

1. Обеспечение функциональности.

Завершенность ПС является общим примитивом качества ПС для выражения и функциональности и надежности ПС, причем для функциональности она является единственным примитивом.

Функциональность ПС определяется его функциональной спецификацией. Завершенность ПС как примитив его качества является мерой того, в какой степени эта спецификация реализована в разрабатываемом ПС. Обеспечение этого примитива в полном объеме означает реализацию каждой из функций, определенной в функциональной спецификации, со всеми указанными там деталями и особенностями. Все рассмотренные ранее технологические процессы показывают, как это может быть сделано.

Однако в спецификации качества ПС могут быть определены несколько уровней реализации функциональности ПС: может быть определена некоторая упрощенная (начальная или стартовая) версия, которая должна быть реализована в первую очередь; могут быть также определены и несколько промежуточных версий. В этом случае возникает дополнительная технологическая задача: организация наращивания функциональности ПС. Здесь важно отметить, что разработка упрощенной версии ПС не есть разработка его прототипа. Прототип разрабатывается для того, чтобы лучше понять условия применения будущего ПС, уточнить его внешнее описание. Он рассчитан на избранных пользователей и поэтому может сильно отличаться от требуемого ПС не только выполняемыми функциями, но и особенностями пользовательского интерфейса. Упрощенная же версия разрабатываемого ПС должна быть рассчитана на практически полезное применение любыми пользователями, для которых предназначено это ПС. Поэтому главный принцип обеспечения функциональности такого ПС заключается в том, чтобы с самого начала разрабатывать ПС таким образом, как будто требуется ПС в полном объеме, до тех пор, когда разработчики будут иметь дело непосредственно с теми частями или деталями ПС, реализацию которых можно отложить в соответствии со спецификацией его качества. Тем самым, и внешнее описание и описание архитектуры ПС должно быть разработано в полном объеме. Можно отложить лишь реализацию тех программных подсистем (определенных в архитектуре разрабатываемого ПС), функционирования которых не требуется в начальной версии этого ПС. Реализацию же самих программных подсистем лучше всего производить методом целенаправленной конструктивной реализации, оставляя в начальной версии ПС подходящие имитаторы тех программных модулей, функционирование которых в этой версии не требуется. Допустима также упрощенная реализация некоторых программных модулей, опускающая реализацию некоторых деталей соответствующих функций. Однако такие модули с технологической точки зрения лучше рассматривать как своеобразные их имитаторы (хотя и далеко продвинутые).

Достигнутый при обеспечении функциональности ПС уровень его завершенности на самом деле может быть не таким, как ожидалось, из-за ошибок, оставшихся в этом ПС. Можно лишь говорить, что требуемая завершенность достигнута с некоторой вероятностью, определяемой объемом и качеством проведенного тестирования. Для того чтобы повысить эту вероятность, необходимо продолжить тестирование и отладку ПС. Однако, оценивание такой вероятности является весьма специфической задачей, которая пока еще ждет соответствующих теоретических исследований.

2. Обеспечение надежности.

Надежность имеющегося в распоряжении разработчиков базового программного обеспечения для целевого компьютера может не отвечать требованиям к надежности разрабатываемого ПС. Поэтому от использования такого программного обеспечения приходиться отказываться, а его функции в требуемом объеме приходится реализовывать в рамках разрабатываемого ПС. Аналогичное решение приходится принимать при жестких ограничениях на используемые ресурсы (по критерию эффективности ПС). Такое решение может быть принято уже в процессе разработки спецификации качества ПС, иногда – на этапе конструирования ПС.

Этот примитив качества ПС обеспечивается с помощью так называемого защитного программирования. Вообще говоря, защитное программирование применяется при программировании модуля для повышения надежности ПС в более широком смысле. Как утверждает Майерс [11.3], “защитное программирование основано на важной предпосылке: худшее, что может сделать модуль, – это принять неправильные входные данные и затем вернуть неверный, но правдоподобный результат”. Для того, чтобы этого избежать, в текст модуля включают проверки его входных и выходных данных на их корректность в соответствии со спецификацией этого модуля, в частности, должны быть проверены выполнение ограничений на входные и выходные данные и соотношений между ними, указанные в спецификации модуля. В случае отрицательного результата проверки возбуждается соответствующая исключительная ситуация. Для обработки таких ситуаций в конец этого модуля включаются фрагменты второго рода – обработчики соответствующих исключительных ситуаций. Эти обработчики помимо выдачи необходимой диагностической информации, могут принять меры либо по исключению ошибки в данных (например, потребовать их повторного ввода), либо по ослаблению влияния ошибки (например, во избежание поломки устройств, управляемых с помощью данного ПС, при аварийном прекращении выполнения программы осуществляют мягкую их остановку).

Применение защитного программирования модулей приводит к снижению эффективности ПС как по времени, так и по памяти. Поэтому необходимо разумно регулировать степень применения защитного программирования в зависимости от требований к надежности и эффективности ПС, сформулированных в спецификации качества разрабатываемого ПС. Входные данные разрабатываемого модуля могут поступать как непосредственно от пользователя, так и от других модулей. Наиболее употребительным случаем применения защитного программирования является применение его для первой группы данных, что и означает реализацию устойчивости ПС. Это нужно делать всегда, когда в спецификации качества ПС имеется требование об обеспечении устойчивости ПС. Применение защитного программирования для второй группы входных данных означает попытку обнаружить ошибку в других модулях во время выполнения разрабатываемого модуля, а для выходных данных разрабатываемого модуля - попытку обнаружить ошибку в самом этом модуле во время его выполнения. По существу, это означает частичное воплощение подхода самообнаружения ошибок для обеспечения надежности ПС, о чем шла речь в лекции 3. Этот случай защитного программирования применяется крайне редко - только в том случае, когда требования к надежности ПС чрезвычайно высоки.

3. Обеспечение качества программного средства.

Спецификация качества определяет основные ориентиры (цели), которые на всех этапах разработки ПС так или иначе влияют при принятии различных решений на выбор подходящего варианта. Однако каждый примитив качества имеет свои особенности такого

влияния, тем самым, обеспечение его наличия в ПС может потребовать своих подходов и методов разработки ПС или отдельных его частей. Кроме того, уже отмечалась ранее противоречивость критериев качества ПС, а также и выражающих их примитивов качества: хорошее обеспечение одного какого-либо примитива качества ПС может существенно затруднить или сделать невозможным обеспечение некоторых других из этих примитивов. Поэтому существенная часть процесса обеспечения качества ПС состоит из поиска приемлемых компромиссов. Эти компромиссы частично должны быть определены уже в спецификации качества ПС: модель качества ПС должна конкретизировать требуемую степень присутствия в ПС каждого его примитива качества и определять приоритеты достижения этих степеней.

Обеспечение качества осуществляется в каждом технологическом процессе: принятые в нем решения в той или иной степени оказывают влияние на качество ПС в целом. В частности и потому, что значительная часть примитивов качества связана не столько со свойствами программ, входящих в ПС, сколько со свойствами документации. В силу отмеченной противоречивости примитивов качества весьма важно придерживаться выбранных приоритетов в их обеспечении. При этом следует придерживаться двух общих принципов:

- сначала необходимо обеспечить требуемую функциональность и надежность ПС, а затем уже доводить остальные критерии качества до приемлемого уровня их присутствия в ПС;
- нет никакой необходимости и, может быть, даже вредно добиваться более высокого уровня присутствия в ПС какого-либо примитива качества, чем тот, который определен в спецификации качества ПС.

Обеспечение функциональности и надежности ПС было рассмотрено в предыдущей лекции. Ниже обсуждается обеспечение других критериев качества ПС.

Легкость применения, в значительной степени, определяется составом и качеством пользовательской документации, а также некоторыми свойствами, реализуемыми программным путем.

С пользовательской документацией связаны такие примитивы качества ПС, как П-документированность и информативность. Обеспечением ее качества занимаются обычно технические писатели. Этот вопрос будет обсуждаться в следующей лекции. Здесь лишь следует заметить, что там речь будет идти об автономной по отношению к программам документации. В связи с этим следует обратить внимание на широко используемый в настоящее время подход информирования пользователя в интерактивном режиме (в процессе применения программ ПС). Такое информирование во многих случаях оказывается более удобным для пользователя, чем с помощью автономной документации, так как позволяет пользователю без какого-либо поиска вызывать необходимую информацию за счет использования контекста ее вызова. Такой подход к информированию пользователя является весьма перспективным.