**Приложение №7**

|  |  |
| --- | --- |
| **Дата** | 15.12.2023 |
| **Тема** | 1. Проектирование программного модуля  2. Объединение спроектированных программных модулей в единое программное решение  3. Определение и выделение необходимых программных модулей для работы программы  4. Разработка программы по выделенным программным модулям  5. Выполнение индивидуального задания варианта 10 (9) |
| **Задания** | 1. Спроектировать программный модуль  2. Объединить спроектированные программные модулей в единое программное решение  3. Определить и выделить необходимые программные модули для работы программы  4. Разработать программу по выделенным программным модулям  5. Выполнение индивидуального задания варианта 10 (9) |

**1. Проектирование программного модуля**

Сложная система обычно может быть разделена на более простые части — модули. Модульность является важным качеством инженерных процессов и продуктов. Большинство промышленных процессов являются модульными и составлены из комплексов работ, которые комбинируются простыми способами (последовательными или перекрывающимися) для достижения требуемого результата. Главное преимущество модульности заключается в том, что она позволяет применять принцип разделения задач на двух этапах:

при работе с элементами каждого модуля отдельно (игнорируя элементы других модулей)

при работе с общими характеристиками групп модулей и отношениями между ними с целью объединить их в конкретный, более крупный и сложный компонент.

Если данные этапы выполняются в последовательности, предусматривающей сначала концентрацию процессов на модулях, а затем — их объединение, то система проектируется снизу вверх. Если сначала систему разбивают на модули, а потом работают над их индивидуальным проектированием, то это — проектирование сверху вниз.

При структурном построении комплексов программ важное значение имеет размер и сложность компонентов для каждого уровня иерархии и соответственно число иерархических уровней для крупных ПС. По принципам построения, языку описания, размеру и другим характеристикам компонентов в структуре ПС можно выделить иерархические уровни:

— программных модулей, оформляемых как законченные компоненты текста программ;

— функциональных групп (компонентов) или пакетов программ;

— комплексов программ, оформляемых как законченные программные продукты определенного целевого назначения.

С повышением иерархического уровня увеличивается размер текста программ, реализующих компоненты этого уровня и количество обрабатываемых переменных. Одновременно совокупности команд все более специализируется и снижается возможность повторного применения компонентов в различных комбинациях для решения аналогичных задач.

Программные модули решают относительно небольшие функциональные задачи, и каждый реализуется 10—100 операторами языка программирования.

Каждый модуль может использовать на входе около десятка типов переменных. Если для решения небольшой функциональной задачи требуется более 100 операторов, то обычно целесообразно проводить декомпозицию задачи на несколько более простых модулей.

Функциональные группы программ (компоненты) формируются на базе нескольких или десятков модулей и решают достаточно сложные автономные задачи. На их реализацию целесообразно использовать до десятка тысяч строк текста программы. Соответственно возрастают число используемых типов переменных и разнообразие выходных данных. При этом быстро растет число типов переменных, обрабатываемых модулями и локализующихся в пределах одного или нескольких модулей.

Комплексы программ — программные продукты создаются для решения сложных задач управления и обработки информации. В комплексы объединяются несколько или десятки функциональных групп программ для решения общей целевой задачи системы. Размеры ПС зачастую исчисляются сотнями модулей, десятками и сотнями тысяч операторов. Встречаются ПС, содержащие до двух-трех десятков структурных иерархических уровней, построенных из модулей.

Проектирование модулей включает в себя разработку локальных функций и подробных описаний алгоритмов обработки данных; межмодульных интерфейсов; внутренних структур данных; структурных схем передач управления; средств управления в исключительных ситуациях. С их помощью определяются функции: порядок следования отдельных шагов обработки, ситуации и типы данных, вызывающие изменения процесса обработки, а также повторно используемые функции программы. Программные модули для их многократного использования должны базироваться на унифицированных правилах структурного построения, оформления спецификаций требований и описаний текстов программ и комментариев.

Кроме того, целесообразно для каждого проекта директивно ограничивать размеры модулей по числу строк текста с учетом языка программирования, например, 30-ю или 50-ю операторами.

**Основная цель такой унификации** — облегчить разработку модулей, обеспечение их качества и тестирования, а также упростить управление их функциями и характеристиками.

**2. Объединение спроектированных программных модулей в единое программное решение**

Интеграция – процесс разработки и внедрения программного обеспечения, с помощью которого отдельные компоненты могут быть связаны в единую систему. Такое объединение позволяет поддерживать бизнес-процессы и оперативно обмениваться информацией.

Главная задача процесса – обеспечение безопасного и бесперебойного обмена информацией между программными продуктами, которые изначально не предназначены для совместной работы. Например, программное обеспечение для электронного документооборота между предприятием и его клиентами, организация цепей поставок, ERP-системы, облачные технологии, аналитические модули, системы самообслуживания и т. д.

С помощью интеграции программных модулей обеспечивается не только оперативность и автоматизация работы с данными. Решается ряд более широких задач:

* оперативное внедрение новых программных продуктов в работу предприятия;
* улучшения качества работы с клиентами;
* прозрачность процессов;
* сокращения количества ошибок при обработке данных и т. д.

Прежде, чем настраивать интеграцию модулей, нужно определиться с принимающей и передающей стороной – источником данных. Если изначально источник информации не определен, сделать это нужно до начала загрузки данных в систему, так как в ней могут присутствовать приложения с идентичным функционалом. Например, ПО 1С: Торговля и любая стандартная CRM позволяют вносить данные контрагентов. Разрозненный ввод повлечет за собой проблемы задвоенных данных, их несвоевременную синхронизацию, потерю и т. д.

Возможна и двусторонняя интеграция. В этом случае источник и принимающая сторона могут меняться.

Одно из главных преимуществ интеграции программных модулей – постоянный доступ к обновляющимся данным. Именно за счет этого удается создавать инструменты и сервисы, которые оперативно адаптируются под конкретные задачи.

Основной принцип работы процессов – взаимодействие модулей друг с другом, каждый из которых выполняет конкретную задачу. При завершении работы одного из компонентов все обработанные данные должны быть оперативно синхронизированы с другими модулями и доступны для дальнейших действий.

Чтобы гармонично объединить все элементы системы, важно придерживаться следующих принципов.

API – комплекс правил, на основании которых взаимодействуют отдельные части ПО. Именно за счет API программистам удается оперативно настроить компоненты, предупредить и устранить возможные ошибки в работе модулей.

Настраивается специальный комплекс действий, которые запускаются после произведения другого, заранее определенного действия. Например, при завершении оформления заказа клиентом система автоматически генерирует ему счет на оплату, то есть одно действие является триггером для последующего.

Правильная настройка обмена данными между разными модулями системы позволяет ускорить обмен и синхронизацию данных.

В зависимости от цели внедрения интеграцию, в первую очередь, подразделяют на внутреннюю и внешнюю. Внутренняя подразумевает добавление конкретных программных модулей без привлечения внешних ресурсов. Внешняя интеграция позволяет внутренние процессы синхронизировать с более глобальными, например, подключение к Google Maps для логистических компаний.

Доступны 3 вида интеграции:

* облачная;
* локальная;
* гибридная.

Каждый из видов, в свою очередь, различается по следующим методам:

* На уровне брокеров. Данный вид интеграции считается универсальным. При необходимости задействуется дополнительный модуль – брокер. Он подключается к другим необходимым модулям. Такой вид интеграции считается сложным в реализации, требует определенных знаний.
* На уровне интерфейсов. Целью данного вида интеграции изначально было объединение разноплановых приложений. Сложность такого типа в последовательном подключении элементов. Это вызывает ряд ошибок в процессе взаимодействия. К тому же часто встречаются Legacy софт.
* На уровне сервисов. Здесь при помощи программного обеспечения осуществляется фиксация данных и интерфейсов с двух сторон. Это один из немногих видов неавтоматизированной интеграции, то есть участие человека здесь остро необходимо.

Функционально-прикладная и организационная интеграция. Ключевым моментом здесь является объединение нескольких схожих или однотипных приложений. Этот вид наиболее удобен для крупных предприятий, корпораций. Именно за счет интеграции этого вида удается снизить затраты на обслуживающий персонал, так как практически все процессы максимально доступны.

Корпоративные программные приложения. Здесь используются не только приложения внутри системы, но и сам исполняемый код. Специализированное ПО и API позволяют использовать отдельные компоненты приложений в единое ядро. Такую систему легче администрировать и масштабировать при необходимости. Доступ к ядру осуществляется при помощи стандартных протоколов доступа, например, SOAP.

**3. Определение и выделение необходимых программных модулей для работы программы**

Модульное программирование является особым способом разработки программы, которая строится при этом из нескольких относительно независимых друг от друга частей – модулей. Понятие модуля является одним из центральных при разработке программного обеспечения.

**Модуль** — функционально законченный фрагмент программы, оформленный в виде отдельного файла с исходным кодом или поименованной непрерывной её части, предназначенный для использования в других программах. Модули позволяют разбивать сложные задачи на более мелкие в соответствии с принципом модульности. Обычно проектируются таким образом, чтобы предоставлять программистам удобную для многократного использования функциональность (интерфейс) в виде набора функций, классов, констант. Модули могут объединяться в пакеты и, далее, в библиотеки. Удобство использования модульной архитектуры заключается в возможности обновления (замены) модуля, без необходимости изменения остальной системы. В большинстве случаев различные модули могут запускаться как на одном сервере, так и на разных, для распределения нагрузки и создания распределенной архитектуры.

**Модульное программирование** — это организация программы как совокупности небольших независимых блоков, называемых модулями, структура и поведение которых подчиняются определенным правилам. Использование модульного программирования позволяет упростить тестирование программы и обнаружение ошибок. Аппаратно-зависимые подзадачи могут быть строго отделены от других подзадач, что улучшает мобильность создаваемых программ.

Суть **модульного программирования** состоит в разбиении сложной задачи на некоторое число более простых подзадач и составлении программ для решения достаточно независимо друг от друга. Модульность является одним из основных принципов построения программных проектов.

Языки программирования, поддерживающие модульный подход, описывают модуль как программную единицу, состоящую из двух основных частей - спецификации (интерфейса) и реализации. В спецификации приводятся такие характеристики объектов модуля, которые необходимы и достаточны для использования этих объектов в других модулях и программах. Это позволяет использовать объекты модулей только на основе информации об их интерфейсе (не ожидая их полного описания). В реализационной части модуля описывается представление и алгоритмы обработки, связанные с теми или иными объектами модуля.

Модуль является одним из средств, облегчающих верификацию программ. Модуль, как средство создания абстракции, выделяет спецификацию и локализует сведения о реализации.

Модули служат также целям создания проблемно-ориентированного контекста и локализации машинной зависимости.

**Концепция модульного программирования**

Концепцию модульного программирования можно сформулировать в виде нескольких понятий и положений:

* Функциональная декомпозиция задачи- разбиение большой задачи на ряд более мелких, функционально самостоятельных подзадач - модулей. Модули связаны между собой только по входным и выходным данным.
* Модуль - основа концепции модульного программирования. Каждый модуль в функциональной декомпозиции представляет собой "черный ящик" с одним входом и одним выходом. Модульный подход позволяет безболезненно производить модернизацию программы в процессе ее эксплуатации и облегчает ее сопровождение. Дополнительно модульный подход позволяет разрабатывать части программ одного проекта на разных языках программирования, после чего с помощью компоновочных средств объединять их в единый загрузочный модуль.
* Реализуемые решения должны быть простыми и ясными. Если назначение модуля непонятно, то это говорит о том, что декомпозиция начальной или промежуточной задачи была проведена недостаточно качественно. В этом случае необходимо еще раз проанализировать задачу и, возможно, провести дополнительное разбиение на подзадачи. При наличии сложных мест в проекте, их нужно подробнее документировать с помощью продуманной системы комментариев. Этот процесс нужно продолжать до тех пор, пока действительно не удастся добиться ясного понимания назначения всех модулей задачи и их оптимального сочетания.
* Назначение всех переменных модуля должно быть описано с помощью комментариев по мере их определения.

**4. Разработка программы по выделенным программным модулям**

В качестве модульной структуры программы принято использовать древовидную структуру, включая деревья со сросшимися ветвями. В узлах дерева размещаются модули, а стрелки показывают подчиненность модулей (т.е. в тексте модуля, из которого она исходит, имеется ссылка на модуль, в который она входит)

Модульная структура программы должна включать и совокупность спецификаций модулей, образующих эту программу. *Спецификация*программного модуля содержит:

* синтаксическую спецификацию его входов, позволяющую построить обращение к нему на используемом языке программирования
* функциональную спецификацию модуля (описание всех функций, выполняемых этим модулем).
* Существуют разные методы разработки структуры программы. Обычно используют 2 метода:
* Метод восходящей разработки
* метод нисходящей разработки

Метод восходящей разработки. Сначала строится модульная структура программы в виде дерева. Затем поочередно программируются модули программы, начиная с модулей самого нижнего уровня, в таком порядке, чтобы для каждого программируемого модуля были уже запрограммированы все модули, к которым он может обращаться. После того, как все модули программы запрограммированы, производится их поочередное тестирование и отладка в принципе в таком же (восходящем) порядке, в каком велось их программирование.

На первый взгляд такой порядок разработки программы кажется вполне естественным: каждый модуль при программировании выражается через уже запрограммированные непосредственно подчиненные модули, а при тестировании использует уже отлаженные модули. Не рекомендуется, т.к.:

* для программирования какого-либо модуля совсем не требуется текстов используемых им модулей - для этого достаточно, чтобы каждый используемый модуль был лишь специфицирован (в объеме, позволяющем построить правильное обращение к нему), а для тестирования его возможно используемые модули заменять их имитаторами (заглушками).
* каждая программа в какой-то степени подчиняется некоторым внутренним для нее, но глобальным для ее модулей соображениям (принципам реализации, предположениям, структурам данных и т.п.), что определяет ее концептуальную целостность и формируется в процессе ее разработки. При восходящей разработке эта глобальная информация для модулей нижних уровней еще не ясна в полном объеме, поэтому очень часто приходится их перепрограммировать.
* при восходящем тестировании для каждого модуля (кроме головного) приходится создавать ведущую программу (модуль), которая должна подготовить для тестируемого модуля необходимое состояние информационной среды и произвести требуемое обращение к нему. Это приводит к большому объему "отладочного" программирования и в то же время не дает никакой гарантии, что тестирование модулей производилось именно в тех условиях, в которых они будут выполняться в рабочей программе.

Метод *нисходящей разработки*.

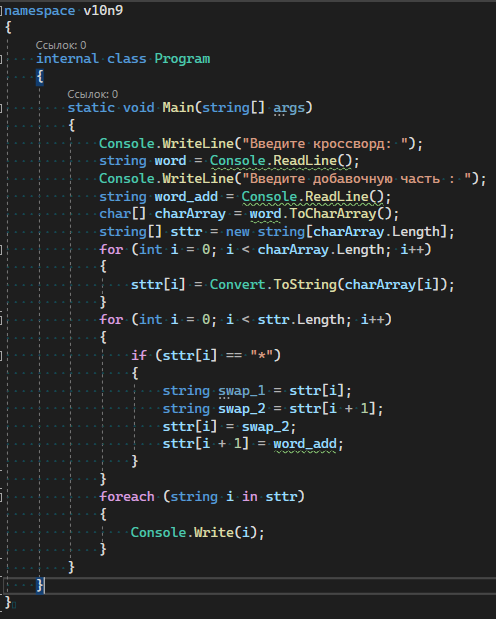
Сначала строится модульная структура программы в виде дерева. Затем поочередно программируются модули программы, начиная с модуля самого верхнего уровня (головного), переходя к программированию какого-либо другого модуля только в том случае, если уже запрограммирован модуль, который к нему обращается. После того, как все модули программы запрограммированы, производится их поочередное тестирование и отладка в таком же (нисходящем) порядке.

Положительные стороны

При таком порядке разработки программы вся необходимая глобальная информация формируется своевременно, т.е. ликвидируется весьма неприятный источник просчетов при программировании модулей.

Существенно облегчается и тестирование модулей, производимое при нисходящем тестировании программы. Первым тестируется головной модуль программы, который представляет всю тестируемую программу и поэтому тестируется при "естественном" состоянии информационной среды, при котором начинает выполняться эта программа. При этом все модули, к которым может обращаться головной, заменяются на их имитаторы. И*митатор модуля -*простой программный фрагмент, сигнализирующий, о самом факте обращения к имитируемому модулю с необходимой для правильной работы программы обработкой значений его входных параметров и с выдачей, если это необходимо, заранее запасенного подходящего результата. После завершения тестирования и отладки головного и любого последующего модуля производится переход к тестированию одного из модулей, которые в данный момент представлены имитаторами, если таковые имеются. Для этого имитатор выбранного для тестирования модуля заменяется на сам этот модуль и добавляются имитаторы тех модулей, к которым может обращаться выбранный для тестирования модуль. При этом каждый такой модуль будет тестироваться при "естественных" состояниях информационной среды, возникающих к моменту обращения к этому модулю при выполнении тестируемой программы. Таким образом большой объем "отладочного" программирования заменяется программированием достаточно простых имитаторов используемых в программе модулей. Кроме того, имитаторы удобно использовать для подыгрывания процессу подбора тестов путем задания нужных результатов, выдаваемых имитаторами.

**5. Выполнение индивидуального задания варианта 10 (9):**



**Выводы:** Спроектировал программный модуль. Объединил спроектированные программные модули в единое программное решение. Определил и выделил необходимые программные модули для работы программы. Разработал программу по выделенным программным модулям. Выполнил индивидуальное задание варианта 10 (9).

Студент: / Ульянов Никита Анатольевич