|  |  |
| --- | --- |
| лого-РГСУ-2015.png | **Российский государственный социальный университет**  **Факультет информационных технологий** |

Практическое задание № 1

**по дисциплине «Технология программирования»**

|  |  |
| --- | --- |
| **ФИО студента** | Тарасова Андрей Валерьевич |
| **Направление подготовки** | Программная инженерия |
| **Группа** | ПИН-2016 |

**Москва 2019**

*Технология структурного программирования*

Структурное программирование - это подход к созданию программ. В результате каждая сложная команда в программе, являющаяся комбинацией последовательных, условных и циклических операторов, имеет только одну точку входа и выхода, что обеспечивает разбиение программы на относительно самостоятельные фрагменты.

Структурное программирование является результатом применения аппликативных методов к императивным программам. Для этого используются процедурно-ориентированные языки, в которых имеется возможность описания программы как совокупности процедур. Процедуры могут вызывать друг друга, и каждая из них может быть вызвана основной программой, которую также можно рассматривать как процедуру.

Структура программы - это искусственно выделенные программистом взаимодействующие части программы. Использование рациональной структуры устраняет проблему сложности разработки, делает программу понятной пользователям, повышает надежность работы программы при сокращении срока ее тестирования и сроков разработки.

Очень часто некоторую последовательность инструкций требуется повторить в разных местах программы. Для того чтобы программисту не приходилось тратить время и усилия на копирование этих инструкций, во многих языках программирования предусматриваются средства для организации подпрограмм. Программист получает возможность присвоить последовательности инструкций произвольное имя и использовать это имя в качестве сокращенной записи в тех местах, где встречается соответствующая последовательность инструкций. Итак, подпрограмма - это некоторая последовательность инструкций, которая может вызываться в нескольких местах программы. Подпрограммы принято называть процедурами и функциями.

Описание подпрограммы состоит из двух частей: заголовки и тела. Заголовок содержит идентификатор подпрограммы и переменные используемые в ней. Тело состоит из одной или нескольких инструкций. Идентификатор подпрограммы используется в качестве сокращенной записи в тех местах программы, где встречается соответствующая последовательность инструкций.

Процедуры и функции, называемые подпрограммами, являются одним из тех немногих фундаментальных инструментов в искусстве программирования, которые оказывают решающее влияние на стиль и качество работы программиста.

Процедура - это не только способ сокращения программного текста, но и, средство разложения программы на логически связанные, замкнутые элементы, определяющие ее структуру. Разложение программы на части существенно для понимания программы, особенно если программа сложная и тяжело обозрима из-за большой длины текста. Разложение на подпрограммы необходимо как для документирования, так и для верификации программы. Поэтому желательно оформлять последовательность инструкций в виде подпрограммы, даже если программа используется однократно и, следовательно, отсутствует мотив, связанный с сокращением текста программы.

Информация о переменных или об условия, которым должны удовлетворять аргументы, задается в заголовке процедуры. О полезности процедуры, в частности, о ее роли при структуризации программы, неоспоримо свидетельствуют еще два понятия в программировании. Вспомогательные или локальные переменные, используемые внутри процедуры, не имеют смысла за ее пределами. В программе существо проще разобраться, если явно указаны области действия таких переменных. Процедура выступает как естественная текстовая единица, с помощью которой ограничивается область существования локальных переменных.

Наиболее общая тактика программирования состоит в разложении процесса на отдельные действия:

- функционального описания на подфункции;

- соответствующие программы на отдельные инструкции.

На каждом таком шаге декомпозиции нужно удостовериться, что решения частных задач приводят к решению общей задачи, выбранная последовательность отдельных действий разумна и позволяет получить инструкции, в каком-либо смысле более близкие к языку, на котором будет реализована программа. Это требование исключает возможность прямолинейного продвижения от первоначальной постановки задачи к конечной программе, которая должна получиться в конечном итоге. Каждый этап декомпозиции сопровождается формулированием частных подпрограмм. В процессе этой работы может обнаружиться, что выбранная декомпозиция неудачна в том смысле хотя бы потому, что подпрограммы неудобно выражать с помощью имеющихся средств. В этом случае один или несколько предыдущих шагов декомпозиции следует пересмотреть заново.

Если видеть в поэтапной декомпозиции и одновременном развитии и детализации программы постепенное продвижение в глубь, то такой метод при решении задач можно охарактеризовать как нисходящий (сверху вниз). И наоборот, возможен такой подход к решению задачи, когда программист сначала изучает имеющиеся в его распоряжении вычислительную машину или язык программирования, затем собирает некоторые последовательности инструкций в элементарные процедуры, типичные для решаемой задачи. Затем элементарные процедуры используются на следующем уровне иерархии процедур. Такой метод перехода от примитивных машинных команд к требуемой реализации программы называется восходящим (снизу вверх).

На практике разработку программы никогда не удается провести строго в одном направлении - сверху вниз или снизу вверх. Однако при конструировании новых алгоритмов нисходящей метод обычно доминирует. С другой стороны, при адаптации программы к несколько измененным требованиям предпочтение зачастую отдается восходящему методу.

Оба метода позволяют разрабатывать программы, которым присуща структура - свойство, отличающее их от аморфных линейных последовательностей инструкций или команд машины. Важно, чтобы используемый язык в полной мере отражал эту структуру. Только тогда окончательный вид полученной программы позволит применить систематические методы верификации.

Если программа разбивается на подпрограммы, то для представления результатов и аргументов часто приходится вводить новые переменные и таким образом устанавливать связь между подпрограммами. Такие переменные следует вводить и описывать на том этапе разработки, на котором они потребовались. Более того, детализация описания процесса может сопровождаться детализацией описания структуры используемых переменных. Следовательно, в языке должны быть средства для отражения иерархической структуры данных. Что показывает, какую важную роль играет при пошаговой разработке программы понятие процедуры, локальности процедур и данных, структурирования данных.

Проектирование начинается с фиксации внешних спецификаций. На основании внешних спецификаций составляется описание внутреннего алгоритма программы, обязательно со структурой внутренних данных. Далее крупные функции разбиваются на подфункции до достижения подфункции размера модуля - подпрограммы языка программирования, к которым предъявляются особые дополнительные требования.

Фундаментальным понятием и функциональным элементом технологии структурного программирования является - модуль.

Модуль - это подпрограмма, но оформленная в соответствии со следующими правилами:

1. модуль должен иметь один вход и один выход и выполнять строго однозначную функцию, которая описывается простым распространенным предложением естественного (русского) языка или даже предложением без сказуемого;

2. модуль должен обеспечивать компиляцию, независимую от других модулей, с «забыванием» всех внутренних значений модулей;

3. модуль может вызывать другие модули по их именам;

4. хороший модуль не использует глобальные переменные для общения с другим модулем, так как потом трудно отыскать модуль, который портит данные. При использовании глобальных переменных, нужно четко комментировать те модули, которые только читают, и те модули, которые могут менять данные;

5. модуль кодируется только стандартными структурами и тщательно комментируется.

В понятие структуры программы включаются состав и описание связей всех модулей, которые реализуют самостоятельные функции программы и описание носителей данных, участвующих в обмене, как между отдельными подпрограммами, так и вводимые и выводимые с внешних устройств.

В случае сложной, большой программы необходимо овладеть специальными приемами получения рациональной структуры программы. Рациональная структура программы обеспечивает почти двукратное сокращение объема программирования и многократное сокращение объемов и сроков тестирования, следовательно, принципиально снижает затраты на разработку.

Подчиненность модулей удобно изображать схемой иерархии. Схема иерархии отражает только подчиненность подпрограмм, но не порядок их вызова или функционирование программы.

До составления схемы иерархии целесообразно составить внешние спецификации программы и составить функциональные описания программы вместе с описанием переменных - носителей данных. Особое внимание надо уделить иерархии типов структурированных данных и их комментированию. Декомпозиция программы на подпрограммы производится по принципу от общего к частному, более детальному. Процесс составления функционального описания и составления схемы иерархии является итерационным. Выбор наилучшего варианта является многокритериальным.

Структурный подход рекомендует соблюдать следующие принципы при создании программного продукта:

- модульность программ;

- структурное кодирование модулей программ;

- нисходящее проектирование рациональной иерархии модулей программ;

- нисходящая реализация программы с использованием заглушек;

- осуществление планирования на всех стадиях проекта;

- сквозной структурный контроль программных комплексов в целом и составляющих их модулей.

Модульность программ характеризуется тем, что вся программа состоит из модулей. Некоторые смысловые группы модулей сосредотачиваются в отдельных файлах. Например, в отдельных файлах (Unit) могут быть сосредоточены модули текстового редактора и модули иерархического меню.

Структурное кодирование модулей программ заключается в особом оформлении их текстов. У модуля должен быть легко различимый заголовок с комментарием, поясняющим функциональное назначение модуля. Имена переменных должны быть мнемоническими. Суть переменных и порядок размещения в них информации должны быть пояснены комментариями, код модуля закодирован с использованием типовых алгоритмических структур с использованием отступов.

Нисходящее проектирование рациональной иерархии модулей программ заключается в выделении первоначальных модулей самого верхнего уровня иерархии, а затем подчиненных модулей.

Нисходящая реализация программы состоит в первичной реализации группы модулей верхних уровней, которые называются ядром программы, и далее постепенно, в соответствии с планом, реализуются модули нижних уровней. Необходимые для линковки программы, недостающие модули имитируются заглушками.

Осуществление планирования на всех стадиях проекта позволяет первоначально спланировать как состав стадий, таки продолжительность всех этапов работ. Такое планирование позволяет завершить разработку в заданный срок при заданных затратах на разработку.

Сквозной структурный контроль заключается в соблюдении заранее намеченного плана тестирования, который охватывает период от разработки внешних спецификаций, далее внутренних спецификаций и их корректировку в периоде реализации вплоть до приемо-сдаточных испытаний. Составляющие программу модули тестируются как во время написания их кода, так и при автономном тестировании, инспекции их исходного кода, при тестировании сразу по подключении к ядру.

При структурном программировании программа в основном реализуется (собирается и тестируется) сверху вниз. Сначала из 20-30 модулей пишется ядро. Чтобы начать тестировать, недостающие модули нижних уровней заменяются заглушками. По окончании тестирования ядра, заглушки заменяются новыми готовыми модулями, но если программа еще не закончена, то для успешной ее линковки понадобятся все новые заглушки недостающих модулей. Затем проводится тестирование собранной части и т. д.

Заглушка - это макет модуля. Самая простая заглушка - это попрограмма или функция без действий. Более сложная заглушка может выводить сообщение о том, что отработал такой-то модуль. Еще более сложные заглушки могут выводить входную информацию в какой-нибудь файл отладки. Наконец, еще более сложные заглушки выдают на выход тестовую информацию, необходимую для проверки уже реализованных модулей. Написание заглушек - лишняя работа, но требуется искусство проектировщика, чтобы максимальное количество заглушек были простыми, а тестирование уже собранной части программы было бы полным.

Проектированию структуры программы предшествует разработка внешних функциональных описаний. Функциональные описания (алгоритмы выполнения программы) для достижения их восприятия должны быть декомпозированы от общего к частному. Также они должны включать описания форм представления и объема внутренних данных.

Для начала имеется первый вариант схемы иерархии, полученный путем простого членения функций программы на подфункции с указанием переменных, необходимых для размещения данных на разных шагах обработки. Скорее этот вариант не является оптимальным, и требуются проектные итерации для улучшения топологии схемы.

Каждый новый вариант сравнивается с предшествующим вариантом по описанным здесь критериям. Генерация вариантов прекращается при невозможности дальнейших улучшений.

Фонд критериев оптимальности схем иерархии является необходимым подспорьем при оптимизации схем иерархии и состоит из следующих критериев:

- полнота выполнения специфицированных функций;

- возможность быстрого и дешевого пополнения новыми, ранее не специфицированными функциями;

- обозримость для проектировщика составных частей программы;

- максимальная независимость по данным отдельных частей программы;

- возможность связывания программы с обширной многоуровневой схемой иерархии конкретным редактором связей (линковщиком). Если начинается работа над новой программой, то очень полезно выполнить на эВМ ее модель в виде пустых заглушек модулей, которые не содержат никаких действий;

- достаточность оперативной памяти. Здесь рассматриваются варианты с описанием особенно структурированных статических и динамических переменных на разных уровнях схемы иерархии. Проверка удовлетворения данного критерия осуществляется расчетами с некоторыми машинными экспериментами;

- оценка влияния топологии схемы иерархии на скорость выполнения программы при использовании оверлеев (динамической загрузки программы) и механизма подкачки страниц при разработке программы, которая целиком не может быть размещена в оперативной памяти;

- отсутствие разных модулей, выполняющих похожие действия. В идеале - один и тот же модуль вызывается на разных уровня схемы иерархии;

- достижение при реализации программы такого сетевого графика работы коллектива программистов, который обеспечивает равномерную загрузку коллектива по ключевым датам проекта;

- всемерно сокращение затрат на тестирование уже собранного ядра программы по ключевым датам сетевого графика реализации. Характеризуется простотой используемых заглушек и качеством тестирования по всем вычислительным маршрутам модулей. Достигается первичной реализацией сверху вниз модулей ввода и вывода программы с отсрочкой реализации остальных ветвей схемы иерархии. Обычно затраты на тестирование составляют около 60% стоимости всего проекта. Хорошая схема иерархии сокращает затраты на тестирование по сравнению с первоначальным вариантом в 2-5 раз и более;

- использованием в данном проекте как можно большего числа разработанных в предшествующих проектах модулей и библиотек при минимальном объеме изготавливаемых заново частей;

- удачное распределение модулей по компилируемым файлам программы и библиотекам;

- накопление уже готовых модулей и библиотек модулей для использования их во всех новых разработках.

В заключении можно отметить что, структурный подход к программированию воспринял и использует многие методы из области проектирования сложных технических систем. Среди них блочно-иерархический подход к проектированию сложных систем, стадийность создания программ, нисходящее проектирование, методы оценки и планирования.

Модуль и основные принципы структурного подхода.

Если программа разбивается на подпрограммы, то для представления результатов и аргументов часто приходится вводить новые переменные и таким образом устанавливать связь между подпрограммами. Такие переменные следует вводить и описывать на том этапе разработки, на котором они потребовались. Более того, детализация описания процесса может сопровождаться детализацией описания структуры используемых переменных. Следовательно, в языке должны быть средства для отражения иерархической структуры данных. Из сказанного видно, какую важную роль играет при пошаговой разработке программы понятие процедуры, локальности процедур и данных, структурирования данных.

Проектирование начинается с фиксации внешних спецификаций. На основании внешних спецификаций составляется описание внутреннего алгоритма программы, обязательно со структурой внутренних данных. Далее крупные функции разбиваются на подфункции до достижения подфункции размера модуля — подпрограммы (процедуры или функции) языка программирования, к которым предъявляются особые дополнительные требования.

*Модуль*— фундаментальное понятие и функциональный элемент технологии структурного программирования.

*Модуль —*это подпрограмма, но оформленная в соответствии с особыми правилами.

*Правило 1.*Модуль должен иметь один вход и один выход и выполнять строго однозначную функцию, которая описывается простым распространенным предложением естественного (русского) языка или даже предложением без сказуемого.

*Правило 2.*Модуль должен обеспечивать компиляцию, независимую от других модулей, с "забыванием" всех внутренних обозначений модулей.

*Правило 3.*Модуль может вызывать другие модули по их именам.

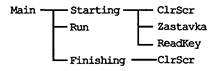
*Правило 4.*Хороший модуль не использует глобальные переменные для общения с другим модулем, так как потом трудно отыскать модуль, который портит данные. Если все же используются глобальные переменные, то нужно четко комментировать те модули, которые только читают, и те модули, которые могут менять данные.

*Правило 5.*Модуль кодируется только стандартными структурами и тщательно комментируется.

В понятие структуры программы включаются состав и описание связей всех модулей, которые реализуют самостоятельные функции программы и описание носителей данных, участвующих в обмене как между отдельными подпрограммами, так и вводимые и выводимые с/на внешних устройств.

В случае сложной, большой программы необходимо овладеть специальными приемами получения рациональной структуры программы. Рациональная структура программы обеспечивает почти двукратное сокращение объема программирования и многократное сокращение объемов и сроков тестирования, а следовательно, принципиально снижает затраты на разработку.

Подчиненность модулей удобно изображать схемой иерархии. Схема иерархии отражает только подчиненность подпрограмм, но не порядок их вызова или функционирование программы. Схема иерархии может иметь вид, показанный на рис. 7.1.



*Рис 7.1.*Схема иерархии простой программы

Такая схема иерархии обычно дополняется расшифровкой функций, выполняемых модулями.

До составления схемы иерархии целесообразно составить внешние спецификации программы и составить функциональные описания программы вместе с описанием переменных-носителей данных. Особое внимание надо уделить иерархии типов структурированных данных и их комментированию. Декомпозиция программы на подпрограммы производится по принципу от общего к частному, более детальному. Вообще процесс составления функционального описания и составления схемы иерархии является итерационным. Выбор наилучшего варианта является многокритериальным.

Расчленение должно обеспечивать удобный порядок ввода частей в эксплуатацию.

Реализация программы (кодирование, сборка, тестирование) должна вестись по разработанному заранее плану и начинаться с верхних модулей схемы иерархии. Недостающие модули нижних уровней заменяются заглушками, которые представляют собой простейшие подпрограммы: либо без действий; либо выводящие в файл отладки входные данные; либо возвращающие в вышестоящие модули тестовые данные (которые обычно присваиваются внутри заглушки); либо содержащие комбинацию этих действий.

Структурный подход к программированию представляет собой как методологию, так и технологию создания программ. В свое время его внедрение обеспечило повышение производительности труда программистов при написании и отладке программ; получение программ, которые состоят из модулей и пригодны для сопровождения; создание программ коллективом разработчиков; окончание создания программы в заданный срок.

Структурный подход к программированию воспринял и использует многие методы из области проектирования сложных технических систем. Среди них блочно-иерархический подход к проектированию сложных систем, стадийность создания программ, нисходящее проектирование, методы оценки и планирования.

Структурный подход рекомендует соблюдать следующие принципы при создании программного изделия:

— модульность программ;

— структурное кодирование модулей программ;

— нисходящее проектирование рациональной иерархии модулей программ;

— нисходящая реализация программы с использованием заглушек;

— осуществление планирования на всех стадиях проекта;

— сквозной структурный контроль программных комплексов в целом и составляющих их модулей.

Модульность программ характеризуется тем, что вся программа состоит из модулей. Некоторые смысловые группы модулей сосредоточиваются в отдельных файлах. Например, в отдельных файлах (Unit) могут быть сосредоточены модули текстового редактора и модули иерархического меню.

Структурное кодирование модулей программ заключается в особом оформлении их текстов. У модуля должен быть легко различимый заголовок с комментарием, поясняющим функциональное назначение модуля. Имена переменных должны быть *мнемоническими.*Суть переменных и порядок размещения в них информации должны быть пояснены комментариями, а код модуля закодирован с использованием типовых алгоритмических структур с использованием отступов.

Нисходящее проектирование рациональной иерархии модулей программ заключается в выделении первоначально модулей самого верхнего уровня иерархии, а затем подчиненных модулей.

Нисходящая реализация программы состоит в первичной реализации группы модулей верхних уровней, которые называются ядром программы, и далее постепенно, в соответствии с планом, реализуются модули нижних уровней. Необходимые для линковки программы, недостающие модули имитируются заглушками.

Осуществление планирования на всех стадиях проекта позволяет первоначально спланировать как состав стадий, так и продолжительность всех этапов работ. Такое планирование позволяет завершить разработку в заданный срок при заданных затратах на разработку. Далее планируются порядок и время интеграции модулей во все расширяющееся ядро. Планируются мероприятия по тестированию программы от ранних до заключительных этапов.

Сквозной структурный контроль заключается в соблюдении заранее намеченного плана тестирования, который охватывает период от разработки внешних спецификаций, далее внутренних спецификаций и их корректировку в периоде реализации вплоть до приемо-сдаточных испытаний. Составляющие программу модули тестируются как во время написания их кода, так и при автономном тестировании, инспекции их исходного кода, при тестировании сразу по подключению к ядру.

7.2.2. Понятие заглушки модуля

При структурном программировании программа в основном реализуется (собирается и тестируется) сверху вниз. Сначала из 20–30 модулей пишется ядро. Чтобы начать тестировать, недостающие модули нижних уровней заменяются заглушками. По окончании тестирования ядра, заглушки заменяются новыми готовыми модулями, но если программа еще не закончена, то для успешной ее линковки понадобятся все новые заглушки недостающих модулей. Теперь можно приступать к тестированию собранной части и т. д.

**Заглушка**— это макет модуля. Самая простая заглушка — это подпрограмма или функция без действий. Более сложная заглушка может выводить сообщение о том, что отработал такой-то модуль. Еще более сложные заглушки могут выводить входную информацию в какой-нибудь файл отладки. Наконец, еще более сложные заглушки выдают на выход тестовую информацию, необходимую для проверки уже реализованных модулей.

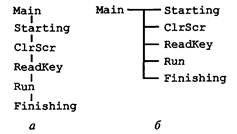
Написание заглушек — "лишняя" работа, но требуется искусство проектировщика, чтобы максимальное количество заглушек были простыми, а тестирование уже собранной части программы было бы полным.

7.2.3. Средства изменения топологии иерархии программы

В процессе нисходящего проектирования рациональной иерархии модулей программы необходимо получить оптимальную подчиненность.

Схеме иерархии можно придать любой топологический рисунок. Так, схеме иерархии, изображенной на рис. 7.2, *а,*можно придать вид, изображенный на рис. 7.2, *б.*

Фрагменты с вертикальными вызовами могут быть преобразованы в вызовы с одного уровня посредством введения дополнительного модуля, который может не выполнять никаких полезных функций с точки зрения алгоритма программы. Функция нового модуля может состоять лишь в мониторинге, т. е. вызове других модулей в определенном порядке.



*Рис. 7.2.* Схема иерархии одной и той же программы: а — с вертикальными вызовами; б — с горизонтальными вызовами

Фрагменты с горизонтальными вызовами на одном уровне могут быть преобразованы в вертикальные вызовы модулей разных уровней посредством введения дополнительных переменных. Эти переменные не могут быть получены путем декомпозиции функционального описания на подфункции. Эти дополнительные переменные обычно имеют тип целый или логический и называются *флагами*(семафорами, ключами) *событий.*Их смысл обычно характеризуется фразой: "В зависимости от предыстории действий, выполнить такие-то действия".

В процессе проектирования нужно сделать несколько проектных итераций, генерируя каждый раз новую схему иерархии, и сравнить эти иерархии по критериям оценки качества схемы иерархии.

7.2.4. Критерии оценки качества схемы иерархии

Проектированию структуры программы предшествует разработка внешних функциональных описаний. Функциональные описания (алгоритмы выполнения программы) для достижения их восприятия должны быть декомпозированы от общего к частному. Также они должны включать описания форм представления и объема внутренних данных.

Итак, сначала имеется первый вариант схемы иерархии, полученный путем простого членения функций программы на подфункции с указанием переменных, необходимых для размещения данных на разных шагах обработки. Скорее этот вариант не является оптимальным, и требуются проектные итерации (обычно выполняются методом "проб и ошибок") для улучшения топологии схемы.

Каждый новый вариант сравнивается с предшествующим вариантом по описанным здесь критериям. Генерация вариантов прекращается при невозможности дальнейших улучшений.

Фонд критериев оптимальности схем иерархии является необходимым подспорьем при оптимизации схем иерархии и состоит из 13 критериев.

*Первый*— полнота выполнения специфицированных функций.

*Второй —*возможность быстрого и дешевого пополнения новыми, ранее не специфицированными функциями.

*Третий,*вытекающий из блочно-иерархического подхода, — обозримость (понятность) для проектировщика составных частей программы.

*Четвертый*критерий оценки качества структуры — максимальная независимость по данным отдельных частей программы.

*Пятый*— возможность связывания программы с обширной многоуровневой схемой иерархии конкретным редактором связей (линковщиком). Если начинаете работать над новой программой, то очень полезно выполнить на ЭВМ ее модель в виде пустых заглушек модулей, которые не содержат никаких действий.

*Шестой*— достаточность оперативной памяти. Здесь рассматриваются варианты с описанием особенно структурированных статических и динамических переменных на разных уровнях схемы иерархии. Проверка удовлетворения данного критерия осуществляется расчетами с некоторыми машинными экспериментами.

*Седьмой*— оценка влияния топологии схемы иерархии на скорость выполнения программы при использовании оверлеев (динамической загрузки программы) и механизма подкачки страниц при разработке программы, которая целиком не может быть размещена в оперативной памяти.

*Восьмой —*отсутствие разных модулей, выполняющих похожие действия. В идеале — один и тот же модуль вызывается на разных уровнях схемы иерархии.

*Девятый*— достижение при реализации программы такого сетевого графика работы коллектива программистов, который обеспечивает равномерную загрузку коллектива по ключевым датам проекта.

*Десятый —*всемерное сокращение затрат на тестирование уже собранного ядра программы по ключевым датам сетевого графика реализации. Характеризуется простотой используемых заглушек и качеством тестирования по всем вычислительным маршрутам модулей. Достигается первичной реализацией сверху вниз модулей ввода и вывода программы с отсрочкой реализации остальных ветвей схемы иерархии. Обычно затраты на тестирование как по срокам, так и деньгам составляют около 60 % стоимости всего проекта. Хорошая схема иерархии сокращает затраты на тестирование по сравнению с первоначальным вариантом в 2–5 раз и более.

*Одиннадцатый*— использование в данном проекте как можно большего числа разработанных в предшествующих проектах модулей и библиотек при минимальном объеме изготавливаемых заново частей.

*Двенадцатый*— удачное распределение модулей по компилируемым файлам программы и библиотекам.

*Тринадцатый —*накопление уже готовых модулей и библиотек модулей для использования их во все новых разработках.

Итак, хорошая структура программы обеспечивает сокращение общего объема текстов в 2 или 3 раза, что соответственно удешевляет проект; на несколько порядков удешевляет тестирование (на тестирование обычно приходится не менее 60 % от общих затрат проекта), а также облегчает и удешевляет сопровождение программы.

7.2.5. Рекомендации по организации процесса разработки схемы иерархии

Как правило, составление внешних, затем внутренних функциональных описаний и далее структуры программы осуществляет группа от двух до семи квалифицированных программистов — системных аналитиков.

Отдельные варианты структуры программы разрабатываются до достижения возможности их сравнения. При этом используются следующие документы:

• описание алгоритма (функционирования) программы или методов решения задачи вместе с описанием данных;

• схема иерархии модулей программы с расшифровкой обозначений и функций модулей;

• паспорта модулей.

На основании этих документов готовятся описание алгоритма программы с учетом модульного деления и сетевой график реализации и тестирования программы с тестами, составленными до программирования.

**Паспорт модуля**— внутренний документ проекта, представляющий собой конверт с именем модуля. Внутри конверта содержатся описания: прототипы вызова самого модуля и модулей, вызываемых модулем данных; расшифровки входных и выходных переменных модуля; описания функции, выполняемой модулем; принципов реализации алгоритма модуля с описанием основных структур данных. В паспорте модуля могут находиться копии литературных источников с описаниями основных идей алгоритма. В процессе выполнения проекта паспорт модуля корректируется до технического задания на разработку этого модуля, а после реализации — до описания модуля.

Группа системных аналитиков проверяет общую однозначность этих описаний и генерирует все новые варианты схем иерархии. При реализации эта группа тестирует уже собранное ядро программы по все новым ключевым датам сетевого графика проекта. В ходе тестирования ядра программы с использованием заглушек уточняются диапазоны данных. В случае необходимости системные аналитики корректируют паспорта модулей перед программированием модулей по результатам уточнения диапазонов данных.

Самое главное в схеме иерархии — минимизация усилий по сборке и тестированию программы. При использовании заглушек можно хорошо тестировать сопряжения модулей, но не сами модули. Тестирование самих модулей потребует изощренных сложных заглушек и астрономического количества тестов. Выход — до интеграции модулей тестировать модули с использованием ведущих программ. Также рекомендуется осуществлять реализацию с некоторым нарушением принципа "сверху — вниз". Рекомендуется сначала с соблюдением принципа "сверху — вниз" реализовать ветвь схемы иерархии, отвечающей за ввод информации с проверкой ее корректности, заглушив ветви расчетов и вывода на самом верхнем уровне. Далее реализуется ветвь вывода информации и в последнюю очередь — ветвь расчетов (функционирования программы). Если функций программы много, то можно сначала реализовать модули выбора функций, заглушив модули самих функций, и далее реализовывать ветвь каждой функции последовательно с соблюдением принципа "сверху — вниз".

Схема иерархии должна включать максимальное количество модулей из других разработок. Многие модули можно использовать в других разработках, однако это не относится к вычислительным модулям, для которых из-за погрешности счета могут не подойти диапазоны данных.

Здесь очень важным является составление удобного графика работы с учетом планирования общего числа кодировщиков программ и их равномерной загрузки по срокам проекта, а также окончание проекта в назначенный срок.

Схема иерархии должна отражаться на файлы с исходными текстами программ таким образом, чтобы каждый файл содержал как можно большее количество готовых функций с общим назначением. Это желательно для облегчения их использования в последующих разработках.

Таким образом, помимо получения денег от заказчика за разработку, программист обязан повышать свой интеллектуальный капитал тоже за деньги заказчика.

Существует очень много автоматизированных систем по формированию декомпозиции схем иерархии, например HIPO, SADT, R-TRAN.

История создания объектно-ориентированного программирования.

Практически сразу после появления языков третьего поколения (1967) ведущие специалисты в области программирования выдвинули идею преобразования постулата фон Неймана: "данные и программы неразличимы в памяти машины". Их цель заключалась в максимальном сближении данных и кода программы. Решая поставленную задачу, они столкнулись с задачей, решить которую без декомпозиции оказалось невозможно, а традиционные структурные декомпозиции не сильно упрощали задачу. Усилия многих программистов и системных аналитиков, направленные на формализацию подхода, увенчались успехом.

Были разработаны три основополагающих принципа того, что потом стало называться объектно-ориентированным программированием (ООПр): наследование; инкапсуляция; полиморфизм.

Результатом их первого применения стал язык Симула-1 (Simula-1), в котором был введен новый тип — объект. В описании этого типа одновременно указывались данные (поля) и процедуры, их обрабатывающие — методы. Родственные объекты объединялись в классы, описания которых оформлялись в виде блоков программы. При этом класс можно использовать в качестве префикса к другим классам, которые становятся в этом случае подклассами первого. Впоследствии Симула-1 был обобщен, и появился первый универсальный ООПр — объектно-ориентированный язык программирования — Симула-67 (67 — по году создания).

Как выяснилось, ООПр оказались пригодными не только для моделирования (Simula) и разработки графических приложений-(SmallTalk), но и для создания большинства других приложений, а их приближенность к человеческому мышлению и возможность многократного использования кода сделали их наиболее используемыми в программировании.

Объектно-ориентированный подход помогает справиться с такими сложными проблемами, как уменьшение сложности программного обеспечения; повышение надежности программного обеспечения; обеспечение возможности модификации отдельных компонентов программного обеспечения без изменения остальных его компонентов; обеспечение возможности повторного использования отдельных компонентов программного обеспечения.

Сравнительный анализ технологий структурного и объектно-ориентированного программирования.

Для проведения сравнительного анализа технологий структурного и объектно-ориентированного программирования разработана специальная методика, основанная на таких объективных принципах, как арифметический подсчет элементов текста программы, анализе алгоритмов программ. Арифметический подсчет выполнялся ручным счетом и был дополнен статистическими данными, выдаваемыми компиляторами и текстовыми редакторами. Итоговые таблицы и их визуализация осуществлялась при помощи программы Excel. Таблицы включают информацию по отдельным файлам и расчет итоговой информации по всей программе.

Информация по отдельным файлам представлена:

1) именем файла;

2) общим количеством строк файла (показывается текстовым редактором);

3) количеством строк операторов описаний данных во всем файле;

4) общим количеством комментариев в файле (выявляется контекстным поиском признака комментария в тексте файла);

5) количеством строк отдельных комментариев в файле;

6) количеством пустых строк в файле (выявляется визуальным анализом текста файла);

7) количеством подпрограмм в файле (является контекстным поиском заголовков procedure и function в тексте файла);

8) количеством операторов описания подпрограмм в файле;

9) количеством строк кода, рассчитанных по формуле: количество строк кода = 2) — 3) — 5) — 6) — 8).

Количество операторов описания подпрограмм в файле выявляется по принципу подсчета всех срок, например, в следующем примере выявлено четыре строки:

function CellString (Col, Row: Word; var Color: Word;

Formatting; Boolean): String;

Begin

End; {CellStrung}

Для проведения объективного сравнительного анализа потребовался выбор функционально похожих программ:

— Mcalc — рассмотренная ранее в гл. 2 и 7 демонстрационная программа, реализованная по технологии структурного программирования;

— Tcalc — демонстрационная программа, реализованная по технологий объектно-ориентированного программирования — функционально полный аналог программы Mcalc.

Результаты арифметического анализа текста программы MCalc, разработанной по технологии структурного программирования, представлены в табл. 8.1.

Таблица 8.1

**Результаты анализа текста программы MCalc**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Имя файла | Всего строк | Количество описательных операторов | Комментарии | | Пустых строк | Количество процедур | Количество описательных операторов процедур | Код |
| Всего | Строк |
| Mcalc | 143 | 8 | 11 | 7 | 5 | 2 | 6 | 117 |
| Mcdisply | 357 | 54 | 47 | 15 | 49 | 18 | 64 | 175 |
| Mcinput | 240 | 33 | 18 | 8 | 19 | 7 | 25 | 155 |
| Mclib | 503 | 68 | 47 | 20 | 46 | 21 | 73 | 296 |
| Mcommand | 873 | 88 | 63 | 19 | 54 | 24 | 86 | 626 |
| Mcparser | 579 | 51 | 33 | 21 | 16 | 12 | 36 | 455 |
| Mcutil | 413 | 62 | 46 | 16 | 45 | 18 | 75 | 215 |
| mcvars | 124 | 96 | 9 | 5 | 19 | 0 | 0 | 0 |
| Итого: | 3232 | 460 | 274 | 111 | 253 | 102 | 365 | 2043 |
|  |  | 15,4% |  | 3,7% |  |  | 12,3% | 68,6% |

Анализ демонстрационный программы TCalc "Borland Inc."

Программа TCalc 1993 (Turbo Pascal 6.0) состоит из следующих файлов:

tcalc.pas — файл основной программы;

tcell.pas — файл работы с клетками;

tcellsp.pas — файл дополнений работы с клетками (изменение значений);

tchash.pas — файл дополнений работы с клетками (значения в клетках);

tcinput.pas — файл подпрограмм ввода данных с клавиатуры;

tclstr.pas — файл подпрограмм работы со строками;

tcmenu.pas — файл подпрограмм, обслуживающих систему меню;

tcparser.pas — файл интерпретатора арифметических выражений формул клеток;

tcrun.pas — файл инициализации и запуска основных объектов;

tcscreen.pas — файл подпрограмм работы с дисплеем;

tcsheet.pas — файл подпрограмм, обслуживающих действия, выбранных посредством меню;

tcutil.pas — файл вспомогательных подпрограмм;

mcmvsmem.asm — ассемблерный файл подпрограмм запоминания в оперативной памяти информации экрана, а также восстановления ранее сохраненной информации экрана.

Все файлы закодированы с соблюдением стандартов оформления.

Хотя фирма "Borland Inc." занимается разработкой компиляторов, файл mcparser.pas также является заимствованным из UNIX YACC utility и лишь частично модифицирован. Остальные файлы являются оригинальными.

Ассемблерный файл mcmvsmem.asm является искусственно добавленным. Цель его добавления — демонстрация возможности использования ассемблерных вставок. Результаты арифметического анализа текста программы представлены в табл. 8.2.

Таблица 8.2

**Результаты анализа текста программы TCalc**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Имя файла | Всего строк | Количество описательных операторов | Комментарии | | Пустых строк | Количество процедур | Количество описательных операторов процедур | Код |
| Всего | Строк |
| Tcalc | 21 | 2 | 9 | 3 | 5 | 1 | 3 | 8 |
| Tcell | 1962 | 490 | 206 | 20 | 153 | 46 | 152 | 1147 |
| Tcellsp | 228 | 39 | 24 | 5 | 18 | 6 | 25 | 141 |
| Tchash | 262 | 50 | 47 | 23 | 23 | 14 | 43 | 123 |
| Tcinput | 334 | 63 | 39 | 15 | 22 | 9 | 32 | 202 |
| Tclstr | 243 | 45 | 120 | 20 | 12 | 15 | 52 | 114 |
| Tcmenu | 234 | 48 | 40 | 20 | 21 | 22 | 66 | 79 |
| Tcparser | 677 | 73 | 29 | 5 | 17 | 9 | 64 | 518 |
| Tcrun | 1367 | 146 | 128 | 59 | 57 | 47 | 163 | 942 |
| Tcscreen | 523 | 215 | 92 | 37 | 16 | 8 | 96 | 159 |
| Tcsheet | 1722 | 240 | 170 | 40 | 44 | 32 | 101 | 1297 |
| Tcutil | 379 | 114 | 55 | 38 | 70 | 29 | 115 | 42 |
| Итого: | 7952 | 1525 | 959 | 285 | 458 | 238 | 912 | 4772 |
|  |  | 20,3% |  | 3,8% |  |  | 12,2% | 63,7% |

В табл. 8.3 и на рис. 8.3 отображены результаты сравнительного анализа технологий структурного и объектно-ориентированного программирования.

Таблица 8.3

**Результаты сравнительного анализа технологий структурного и объектно-ориентированного программирования**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Имя программы | Всего строк | Количество описательных операторов | Комментарии | | Пустых строк | Количество процедур | Количество операторов процедур | Код |
| Всего | Строк |
| MCalc | 3232 | 460 | 274 | 111 | 253 | 102 | 365 | 2043 |
|  |  | 15,4% |  | 3,7% |  |  | 12,3% | 68,6% |
| TCalc | 7952 | 1525 | 959 | 285 | 458 | 238 | 912 | 4772 |
|  |  | 20,3% |  | 3,8% |  |  | 12,2% | 63,7% |

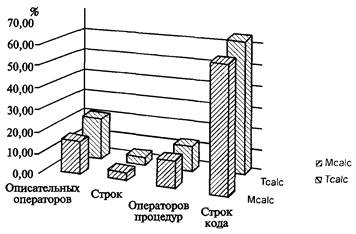
Сравнительный анализ технологий структурного и объектно-ориентированного программирования установил, что для этих технологий наблюдается практически полное совпадение:

• процентного состава описательных операторов;

• процентного состава количества комментариев;

• процентного состава описательных операторов процедур;

• процентного состава операторов кода программы.



*Рис. 8.3.*Результаты сравнительного анализа технологий структурного и объектно-ориентированного программирования

При проведении разработки по технологии объектно-ориентированного программирования по сравнению с технологией структурного программирования объем кода увеличился в 2,34 раза с учетом только кода, выполняющего одни и те же функции (для этого был исключен код функций, аналогичных функциям работы с clipboard Windows). Общее число строк увеличилось в 2,46 раза. Во столько и даже более раз увеличилась трудоемкость разработки.

Собственно функционально полезный код программ Mcalc и Tcalc — одинаковый и составляет порядка 1500 строк.

Почти 2,3–3,5 кратное увеличение трудоемкости разработки объясняется платой за организацию самостоятельности поведения объектов и их завершенную функциональность для повторного использования.