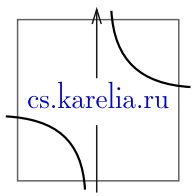
Петрозаводский государственный университет
Институт Математики и информационных технологий
Кафедра Информатики и математического обеспечения

Проектирование программных систем. Интерфейсы и модульность.

Лектор:

к.т.н., доцент Богоявленский Ю. А.
ybgv@cs.karelia.ru



Разделы

- Процесс проектирования интерфейсов
- Декомпозиция подсистем на модули. Модульность.Информационная закрытость, связность
- Типы связности и определение ее уровня

Процесс проектирования интерфейсов
Шаг 1. Знакомство с пользователем.

Уровень знаний и опыт:

- Высокий
- Средний
- Низкий

Факторы:

1. Компьютерная грамотность (низкий – требуется объяснить каждый термин)
2. Системный опыт (низкий – представить примеры и анимацию)
3. Опыт работы с подобными программами
4. Образование
5. Уровень чтения т.д.

Процесс проектирования интерфейсов
Шаг 2. Понимание назначения.

От дизайнера требуется понять цель конкретного предлагаемого пользовательского интерфейса в свете общего назначения программы.

Пример

Цель: инвентаризация склада.
Следствие: пользовательский интерфейс отражает план склада.
Последовательность экранов отражает способ, которым обычно пользователи выполняют свою работу.

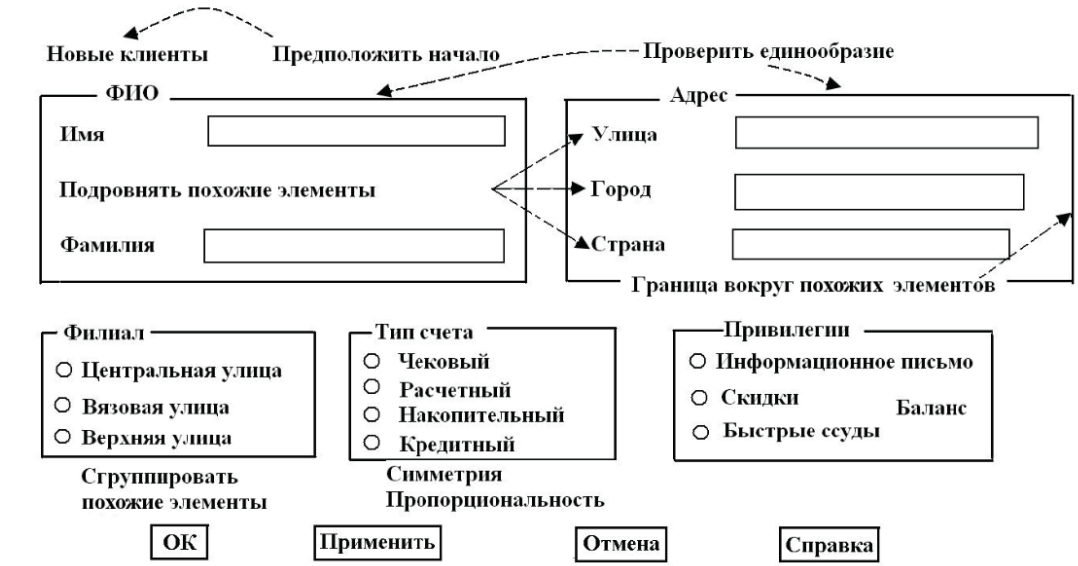
Процесс проектирования интерфейсов
Шаг 3. Выбор экранного дизайна.

Основные прицины хорошего дизайна:

1. Единообразие экранов приложения, а также в логичность каждого отдельно.
2. Предположение о том, откуда пользователь будет начинать работу (“первый” элемент размещают в верхнем левом углу).
3. Навигация должна быть как можно более простой:
   - выровнять похожие элементы;
   - сгруппировать похожие элементы;
   - учесть границы вокруг похожих элементов.

Процесс проектирования интерфейсов

Шаг 3. Пример.



Процесс проектирования интерфейсов

Шаг 4. Выбор подходящего типа окна.

Цели каждого пользовательского интерфейса могут обслуживаться наиболее эффективно одним или двумя конкретными типами окон. Существует пять основных целей графического пользовательского интерфейса:

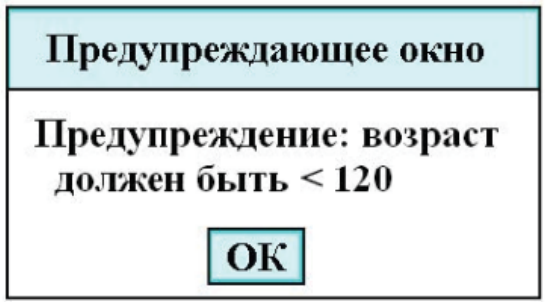
1. Показать свойства объекта

Свойства автомобиля	
Свойство	Значение
Марка	Toyota
Модель	Camry
Номер	893-8913-789014

2. Получить дополнительную информацию для выполнения конкретного задания или команды

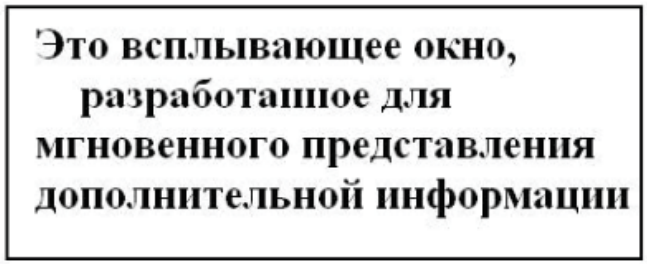
Процесс проектирования интерфейсов
Шаг 4. Выбор подходящего типа окна.

3. Представить информацию



4. Предоставить набор элементов управления

5. Предоставить дополнительную информацию



Процесс проектирования интерфейсов
Шаг 5. Разработка системного меню.

Правила для создания главного меню:

1. сделать главное меню;
2. показать все уместные альтернативы;
3. привести структуру меню в соответствие со структурой задачи приложения;
4. минимизировать число уровней меню.

Процесс проектирования интерфейсов
Шаги 6-7. Выбор подходящих устройств и элементов управления.

Устройства управления

Под устройствами управления понимаются физические устройства, с помощью которых пользователи сообщают свои пожелания приложению. Сюда относятся джойстики, трекболы, графические планшеты, сенсорные экраны, мыши, микрофоны, клавиатуры и др.

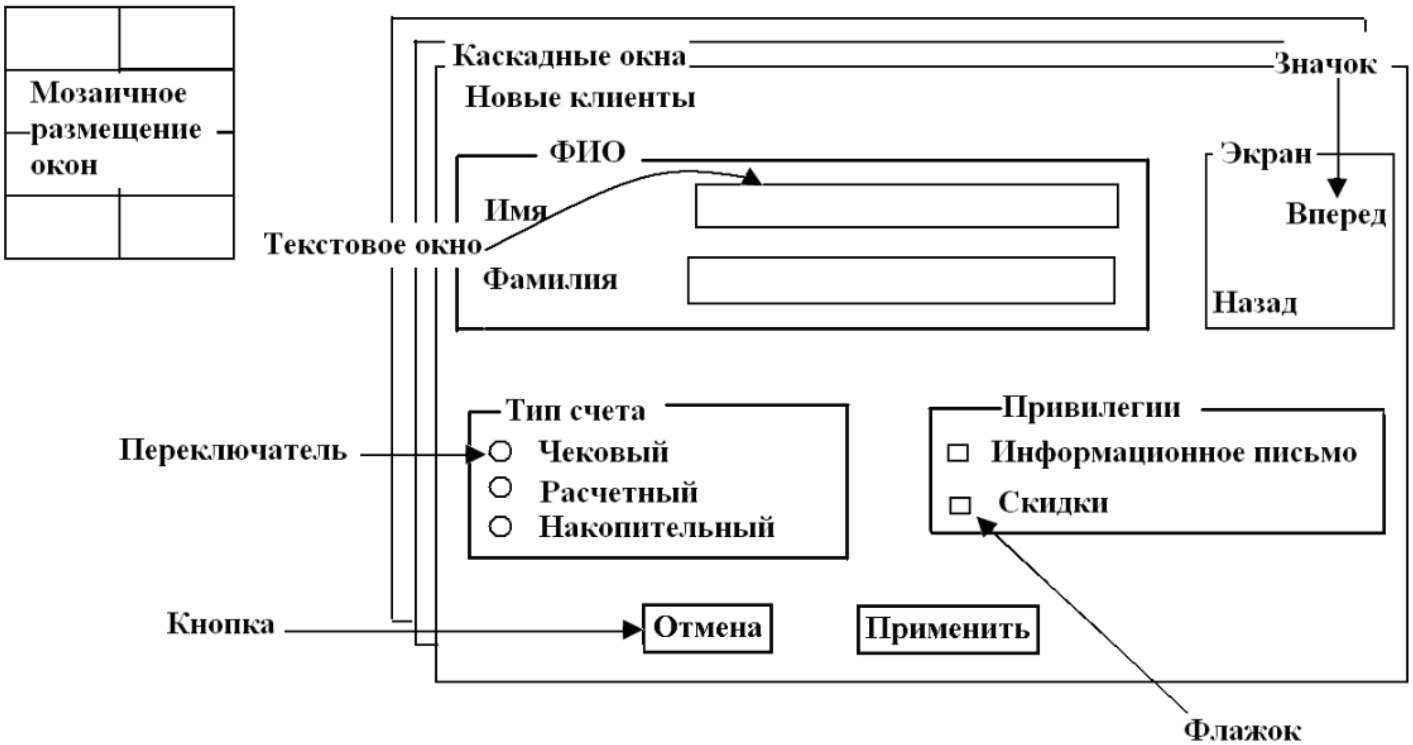
Экранные элементы управления

Экранные элементы управления – это символы, появляющиеся на мониторе, с помощью которых пользователь передает программе вводимые данные и свои намерения.
Сюда относятся значки, кнопки, текстовые окна, списки, изображенные на слайде.

Процесс проектирования интерфейсов
Шаги 8. Организация и планирование окон.

Организация окон:

- соприкасающиеся,
- каскадные.



Процесс проектирования интерфейсов
Шаги 9-11

Выбор подходящих цветов

Если цвет выбран с умением и со вкусом, то это может обогатить экран. Использование цвета не делает автоматически пользовательский интерфейс более полезным или привлекательным, однако может легко его испортить. Инженеры-программисты должны быть очень умеренными и консервативными в использовании цветов.

Осмысленные значки

Значки в интерфейсе должны быть продуманы как в качественном, так и в количественном соотношении.

Сообщения, обратная связь, руководства

Сообщения пользователю должны быть краткими и понятными. Пишется руководство по раОВботе с программным проектом.

Декомпозиция подсистем на модули. Модульность.Информационная закрытость, связность.

Содержание

Введение

После проектирования системной структуры и определения принципов управления структурой следует выполнить разделение подсистем на модули. Фактически эту работу можно считать введением в детальное проектирование, которое конкретизирует архитектурные решения.

Задача декомпозиции — это задача определения внутреннего содержания каждой подсистемы (компонента). Результатом ее решения является формирование **структуры подсистемы** — набора модулей и отношений их взаимодействия.

Известны два подхода и два типа моделей для декомпозиции подсистем на модули:

1. модель потока данных
2. объектно-ориентированная модель

Описание моделей

Модель потока данных

В основе модели потока данных лежит разбиение по функциям. При выборе этой модели получим набор функциональных модулей и определим связи между ними. Например, результат может быть похож на реализацию диаграммы потоков данных.

Объектно-ориентированная модель

Объектно-ориентированная модель основана на объектах (слабо сцепленных сущностях, имеющих собственные наборы данных, состояния, наборы операций) и их описаниях — классах.

Модульность

Модульность — это наиболее общая демонстрация разделения понятий. Программная система делится на именуемые и адресуемые компоненты, часто называемые модулями, которые затем интегрируются для совместного решения проблемы.

Модуль — фрагмент программного текста, являющийся строительным блоком для физической структуры системы. Как правило, модуль состоит из интерфейсной части и части-реализации.

Модульность — свойство системы, которая может подвергаться декомпозиции на ряд внутренне связанных и слабо зависящих друг от друга модулей.

Модульность

Разбиение ПО

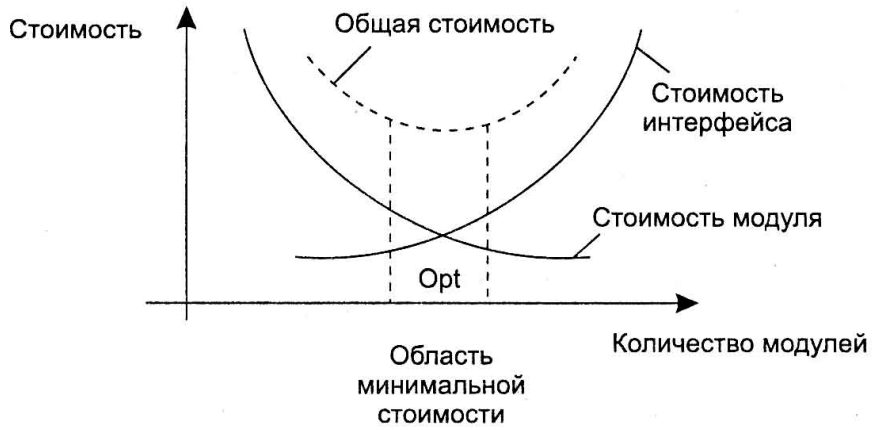
Монолитное ПО, то есть состоящее из одного модуля, не поддается восприятию программным инженером. Огромное количество альтернативных ветвей, множество ссылок, переменных, наконец, общая сложность во многих случаях недоступны для понимания. Требуется разбивать «монолит» на модули в расчете на упрощение понимания и, как следствие, на уменьшение стоимости создания ПО.

Можно прийти к выводу, что путем последовательного дробления ПО можно добиться, что затраты на разработку станут ничтожно малы. Однако не следует забывать про затраты на дальнейшую интеграцию модулей: с увеличением количества модулей такие затраты также растут.

Модульность

Разбиение ПО

Таким образом, существует оптимальное количество модулей Opt, которое приводит к минимальной стоимости разработки.



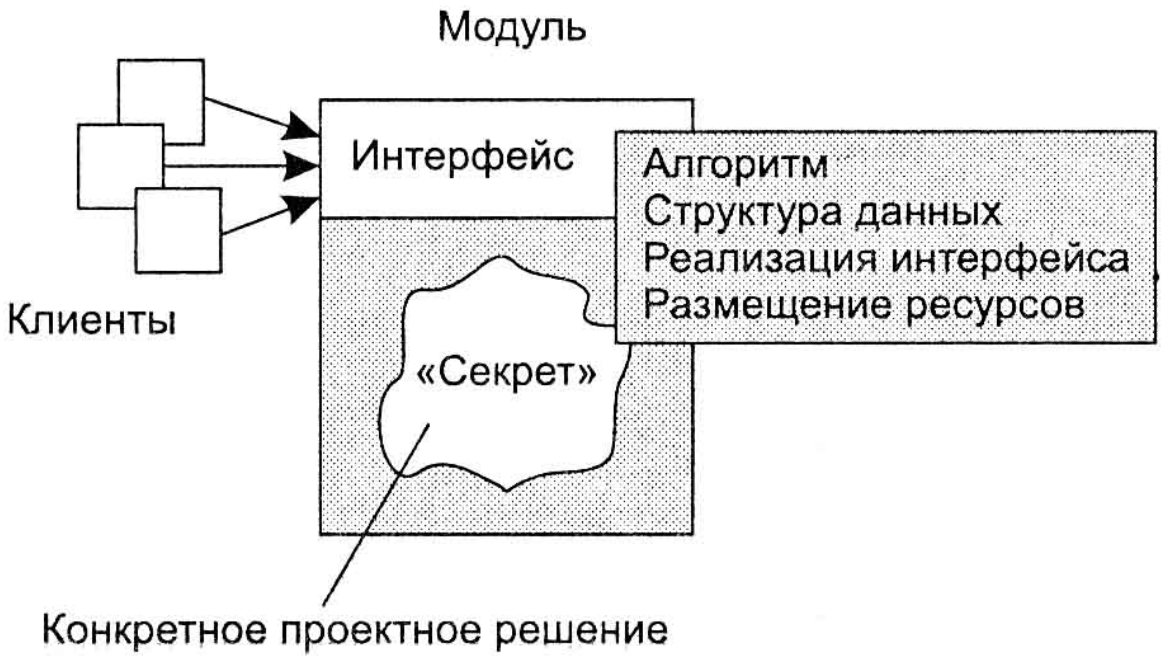
Оптимальный модуль должен удовлетворять двум критериям:

- снаружи он проще, чем внутри
- его проще использовать, чем построить

Информационная закрытость

Принцип **информационной закрытости** утверждает: содержание модулей должно быть скрыто друг от друга.

Модуль должен определяться и проектироваться так, чтобы его содержимое было недоступно тем модулям, которые не нуждаются в такой информации (**клиентам**).



Информационная закрытость

Информационная закрытость означает

- все модули независимы, обмениваются только информацией, необходимой для работы
- доступ к операциям и структурам данных модуля ограничен

Достоинства информационной закрытости

- обеспечивается возможность разработки модулей различными независимыми коллективами
- обеспечивается легкая модификация системы (вероятность распространения ошибок очень мала, так как большинство данных и процедур скрыто от других частей системы).

Связность модуля

Идеальный модуль играет роль «черного ящика», содержимое которого невидимо клиентам. Он прост в использовании — количество «ручек и органов управления» им невелико. Его легко развивать и корректировать в процессе сопровождения программной системы.

Связность модуля — это мера зависимости его частей.
Связность — внутренняя характеристика модуля. Чем выше связность модуля, тем лучше результат проектирования, то есть тем «черней» его ящик, тем меньше «ручек управления» на нем находится и тем проще эти «ручки».

Для измерения связности используют понятие **силы связности** (CC). Существует 7 типов связности.

Связность модуля

Типы связности

1. **Связность по совпадению** ($CC = 0$). В модуле отсутствуют явно выраженные внутренние связи. Например, разработчик спешил на обед, вот и забросил в капсулу модуля все элементы, которые были у него на столе.
2. **Логическая связность** ($CC = 1$). Части модуля объединены по принципу функционального подобия. Например, модуль состоит из разных подпрограмм обработки ошибок. При использовании такого модуля клиент выбирает только одну из подпрограмм. Недостатки:
   - сложное сопряжение
   - большая вероятность внесения ошибок при изменении сопряжения ради одной из функций

Связность модуля

Типы связности

3. **Временна́я связность** ($CC = 3$). Части модуля не связаны, но необходимы в один и тот же период работы системы. Например, в модуле «Утро» могут быть элементы «умыться», «одеться», «позавтракать».
   Недостаток: сильная взаимная связь с другими модулями, отсюда — сильная чувствительность к внесению изменений.
4. **Процедурная связность** ($CC = 5$). Части модуля связаны порядком выполняемых ими действий, реализующих некоторый сценарий поведения. Например, в модуле «Одеваться» могут быть элементы «надеть нижнее белье», «надеть верхнюю одежду».

Связность модуля

Типы связности

5. **Коммуникативная связность** (CC = 7). Части модуля связаны по данным (работают с одной и той же структурой данных). Например, в модуле «Анализ текста» могут быть элементы «подсчитать количество гласных», «подсчитать количество согласных», «подсчитать количество слов».
6. **Информационная (последовательная) связность** (CC = 9). Выходные данные одной части используются как входные данные в другой части модуля. Например, в модуле «Обработка массива» могут быть элементы «инициировать массив», «упорядочить массив», «распечатать массив».
7. **Функциональная связность** (CC = 10). Части модуля вместе реализуют одну функцию. Функция может быть предельно простой, может быть сложной, то есть распадаться на многие части, но с точки зрения внешнего клиента — это всегда единое действие.

Заключение

- Разбиение монолитного ПО на модули позволяет упростить понимание кода и снизить стоимость разработки.
- Увеличение количества модулей повышает стоимость их интеграции.
- Модуль должен определяться и проектироваться так, чтобы его содержимое было недоступно тем модулям, которые не нуждаются в такой информации.
- Чем выше связность модуля, тем проще он для использования и развития.

Типы связности и определение ее уровня. Содержание

- Введение.
- Типы связностей.
- Функциональная связность.
- Информационная связность.
- Коммуникотивная связность.
- Аналогия между информационной и коммуникативной связностью.
- Процедурная связность.
- Временна́я связность.
- Анализ "похожести"процедурной и временно́й связности.
- Логическая связность.
- Связность по совпадению.
- Определение уровня связности.

Введение

Связность модуля (Cohesion) — это мера зависимости его частей. Связность - внутренняя характеристика модуля. Чем выше связность модуля, тем лучше результат проектирования, то есть тем «черней» его ящик (капсула, защитная оболочка модуля), тем меньше «ручек управления» на нем находится и тем проще эти «ручки».

Для измерения связности используют понятие силы связности (CC).

Типы связности

Существует 7 типов связности:

1. Связность по совпадению (CC = 0). В модуле отсутствуют явно выраженные внутренние связи. Например, разработчик спешил на обед, вот и забросил в капсулу модуля все элементы, которые были у него на столе.

2. Логическая связность (CC =1). Части модуля объединены по принципу функционального подобия. Например, модуль состоит из разных подпрограмм обработки ошибок. При использовании такого модуля клиент выбирает только одну из подпрограмм.

Недостатки:

- сложное сопряжение;
- большая вероятность внесения ошибок при изменении сопряжения ради одной из функций.

Типы связности

3. Временна́я связность (CC = 3). Части модуля не связаны, но необходимы в один и тот же период работы системы. Например, в модуле «Утро» могут быть элементы «умыться», «одеться», «позавтракать».

Недостаток: сильная взаимная связь с другими модулями, отсюда — сильная чувствительность к внесению изменений.

4. Процедурная связность (CC = 5). Части модуля связаны порядком выполняемых ими действий, реализующих некоторый сценарий поведения. Например, в модуле «Одеваться» могут быть элементы «надеть нижнее белье», «надеть верхнюю одежду».

5. Коммуникативная связность (CC = 7). Части модуля связаны по данным (работают с одной и той же структурой данных). Например, в модуле «Анализ текста» могут быть элементы «подсчитать количество гласных», «подсчитать количество согласных», «подсчитать количество слов».

Типы связности

6. Информационная (последовательная) связность (CC = 9). Выходные данные одной части используются как входные данные в другой части модуля. Например, в модуле «Обработка массива» могут быть элементы «инициировать массив», «упорядочить массив», «распечатать массив».

7. Функциональная связность (CC = 10). Части модуля вместе реализуют одну функцию. Функция может быть предельно простой, может быть сложной, то есть распадаться на многие части, но с точки зрения внешнего клиента — это всегда единое действие.

Отметим, что типы связности 1-3 — результат неправильного планирования проектного решения, а тип связности 4 — результат небрежного планирования проектного решения приложения.

Характеристика типов связности

Общая характеристика типов связности представлена в следующей таблице

Тип связности	Сопровождаемость	Роль модуля
Функциональная	Лучшая	“Черный ящик”
Информанионная (последовательная)	Лучшая	Не совсем “Черный ящик”
Коммуникативная	Лучшая	“Серый ящик”
Процедурная	Худшая	“Белый” или “просвечивающийся ящик”
Временна́я	Худшая	“Белый ящик”
Лгическая	Худшая	“Белый ящик”
По совпадению	Худшая	“Белый ящик”

Функциональная связность

Функционально связный модуль содержит элементы, участвующие в выполнении одной и только одной проблемной задачи. Примеры функционально связных модулей:

- Вычислять синус угла;
- Проверять орфографию;
- Читать запись файла;
- Вычислять координаты цели;
- Вычислять зарплату сотрудника;
- Определять место пассажира.

Функциональная связность

Некоторые из функционально связных модулей очень просты (например, Вычислять синус угла или Читать запись файла), другие сложны (например, Вычислять координаты цели). Модуль Вычислять синус угла, очевидно, реализует единичную функцию. Несмотря на сложность модуля и на то, что его обязанность исполняют несколько подфункций, если его действия можно представить как единую проблемную функцию (с точки зрения клиента), тогда считают, что модуль функционально связен.

Функциональная связность

Приложения, построенные из функционально связных модулей, легче всего сопровождать. Соблазнительно думать, что любой модуль можно рассматривать как однофункциональный, но не надо заблуждаться. Существует много разновидностей модулей, которые выполняют для клиентов перечень различных работ, и этот перечень нельзя рассматривать как единую проблемную функцию. Критерий при определении уровня связности этих нефункциональных модулей — как связаны друг с другом различные действия, которые они исполняют.

Информационная связность

При информационной (последовательной) связности элементы-обработчики модуля образуют конвейер для обработки данных — результаты одного обработчика используются как исходные данные для следующего обработчика. Приведем пример:

Модуль Прием и проверка записи

прочитать запись из файла
проверить контрольные данные в записи
удалить контрольные поля в записи
вернуть обработанную запись

Конец модуля

Информационная связность

В этом модуле три элемента. Результаты первого элемента (прочитать запись из файла) используются как входные данные для второго элемента (проверить контрольные данные в записи) и т. д. Сопровождать модули с информационной связностью почти так же легко, как и функционально связные модули. Правда, возможности повторного использования здесь ниже, чем в случае функциональной связности. Причина — совместное применение действий модуля с информационной связностью полезно далеко не всегда.

Коммуникативная связность

При коммуникативной связности элементы-обработчики модуля используют одни и те же данные, например внешние данные. Пример коммуникативно связного модуля:

Модуль Отчет и средняя зарплата

используется Таблица зарплаты служащих
сгенерировать Отчет по зарплате
вычислить параметр Средняя зарплата
вернуть Отчет по зарплате, Средняя зарплата

Конец модуля

Здесь все элементы модуля работают со структурой Таблица зарплаты служащих.

Коммуникативная связность

С точки зрения клиента проблема применения коммуникативно связного модуля состоит в избыточности получаемых результатов. Например, клиенту требуется только отчет по зарплате, он не нуждается в значении средней зарплаты. Такой клиент будет вынужден выполнять избыточную работу — выделение в полученных данных материала отчета. Почти всегда разбиение коммуникативно связного модуля на отдельные, функционально связные модули улучшает сопровождаемость системы.

Аналогия между информационной и коммуникативной связностью

Модули с коммуникативной и информационной связностью подобны в том, что содержат элементы, связанные по данным. Их удобно использовать, потому что лишь немногие элементы в этих модулях связаны с внешней средой. Главное различие между ними — информационно связный модуль работает подобно сборочной линии; его обработчики действуют в определенном порядке; в коммуникативно связном модуле порядок выполнения действий безразличен. В нашем примере не имеет значения, когда генерируется отчет (до, после или одновременно с вычислением средней зарплаты).

Процедурная связность

При достижении процедурной связности мы попадаем в пограничную область между хорошей сопровождаемостью (для модулей с более высокими уровнями связности) и плохой сопровождаемостью (для модулей с более низкими уровнями связности). Процедурно связный модуль состоит из элементов, реализующих независимые действия, для которых задан порядок работы, то есть порядок передачи управления. Зависимости по данным между элементами нет.

Процедурная связность

Например:

Модуль Вычисление средних значений

используется Таблица-A, Таблица-B
вычислить среднее по Таблица-A
вычислить среднее по Таблица-B
вернуть среднееТабл-A, среднееТабл-B

Конец модуля

Этот модуль вычисляет средние значения для двух полностью несвязанных таблиц Таблица-A и Таблица-B, каждая из которых имеет по 300 элементов.

Временна́я связность

При связности по времени элементы-обработчики модуля привязаны к конкретному периоду времени (из жизни программной системы). Классическим примером временно́й связности является модуль инициализации:

Модуль Инициализировать Систему

перемотать магнитную ленту 1
Счетчик магнитной ленты 1 := 0
перемотать магнитную ленту 2
Счетчик магнитной ленты 2 := 0
Таблица текущих записей := пробел..пробел
Таблица количества записей := 0..0
Переключатель 1 := выкл
Переключатель 2 := вкл
Конец модуля

Временна́я связность

Элементы данного модуля почти не связаны друг с другом (за исключением того, что должны выполняться в определенное время). Они все — часть программы запуска системы. Зато элементы более тесно взаимодействуют с другими модулями, что приводит к сложным внешним связям.

Модуль со связностью по времени испытывает те же трудности, что и процедурно связный модуль. Программист соблазняется возможностью совместного использования кода (действиями, которые связаны только по времени), модуль становится трудно использовать повторно.

Анализ "похожести"процедурной и временнóй связности

Процедурно связные модули и модули с временнóй связностью очень похожи. Степень их непрозрачности изменяется от темно-серого до светло-серого цвета, так как трудно объявить функцию такого модуля без перечисления ее внутренних деталей. Различие между ними подобно различию между информационной и коммуникативной связностью. Порядок выполнения действий более важен в процедурно связных модулях. Кроме того, процедурные модули имеют тенденцию к совместному использованию циклов и ветвлений, а модули с временнóй связностью чаще содержат более линейный код.

Логическая связность

Элементы логически связного модуля принадлежат к действиям одной категории, и из этой категории клиент выбирает выполняемое действие. Рассмотрим следующий пример:

Модуль Пересылка сообщения

переслать по электронной почте
переслать по факсу
послать в телеконференцию
переслать по ftp-протоколу

Конец модуля

Логическая связность

Как видим, логически связный модуль — мешок доступных действий. Действия вынуждены совместно использовать один и тот же интерфейс модуля. В строке вызова модуля значение каждого параметра зависит от используемого действия. При вызове отдельных действий некоторые параметры должны иметь значение пробела, нулевые значения и т. д. (хотя клиент все же должен использовать их и знать их типы).

Логическая связность

Действия в логически связном модуле попадают в одну категорию, хотя имеют е только сходства, но и различия. К сожалению, это заставляет программиста завязывать код действий в узел», ориентируясь на то, что действия совместно .спользуют общие строки кода. Поэтому логически связный модуль имеет:

- уродливый внешний вид с различными параметрами, обеспечивающими, например, четыре вида доступа;
- запутанную внутреннюю структуру со множеством переходов, похожую на волшебный лабиринт.

В итоге модуль становится сложным как для понимания, так и для сопровождения.

Связность по совпадению

Элементы связного по совпадению модуля вообще не имеют никаких отношений друг с другом:

Модуль Разные функции (какие-то параметры)

поздравить с Новым годом (...)
проверить исправность аппаратуры (...)
заполнить анкету героя (...)
измерить температуру (...)
вывести собаку на прогулку (...)
запастись продуктами (...)
приобрести Ягуар (...)
Конец модуля

Связность по совпадению

Связный по совпадению модуль похож на логически связный модуль. Его элементы-действия не связаны ни потоком данных, ни потоком управления. Но в логически связном модуле действия, по крайней мере, относятся к одной категории; в связном по совпадению модуле даже это не так. Словом, связные по совпадению модули имеют все недостатки логически связных модулей и даже усиливают их. Применение таких модулей вселяет ужас, поскольку один параметр используется для разных целей.

Определение уровня связности

Приведем алгоритм определения уровня связности модуля.

1. Если модуль — единичная проблемно-ориентированная функция, то уровень связности — функциональный; конец алгоритма. В противном случае перейти к пункту 2.

2. Если действия внутри модуля связаны, то перейти к пункту 3. Если действия внутри модуля никак не связаны, то перейти к пункту 6.

3. Если действия внутри модуля связаны данными, то перейти к пункту 4. Если действия внутри модуля связаны потоком управления, перейти к пункту 3.

4. Если порядок действий внутри модуля важен, то уровень связности - информационный. В противном случае уровень связности — коммуникативный. Конец алгоритма.

Определение уровня связности

5. Если порядок действий внутри модуля важен, то уровень связности — процедурный. В противном случае уровень связности — временно́й. Конец алгоритма.

6. Если действия внутри модуля принадлежат к одной категории, то уровень связности — логический. Если действия внутри модуля не принадлежат к одной категории, то уровень связности — по совпадению. Конец алгоритма.

Возможны более сложные случаи, когда с модулем ассоциируются несколько уровней связности.

Определение уровня связности

В этих случаях следует применять одно из двух правил:

- правило параллельной цепи. Если все действия модуля имеют несколько уровней связности, то модулю присваивают самый сильный уровень связности;
- правило последовательной цепи. Если действия в модуле имеют разные уровни связности, то модулю присваивают самый слабый уровень связности.

Спасибо за внимание.