## МЕТРИКИ НА ОСНОВЕ ФУНКЦИОНАЛЬНЫХ УКАЗАТЕЛЕЙ

Метрика дефектов качества:

$$DQ = \frac{\text{Количество\_Ошибок (единиц)}}{\text{Количество\_Строк\_Программы (тыс. строк)}}. \quad (40)$$

Норма составляет $0 \leq DQ \leq 1$.

Количество строк программы часто именуется термином *LOC* (Lines Of Code).

Данную метрику качества можно рассчитать не только на основе регистрации и подсчета ошибок, а так же на основе функциональных указателей (*FP* – Function Points).

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

# Тема 3.4. Процедурно-ориентированные метрики

В качестве процедурно-ориентированных метрик рассматриваются:

– метрики на основе функциональных указателей;
– метрики связности модулей;
– метрики сцепления модулей.

Метрики на основе функциональных указателей представлены:

– метрикой дефектов качества;
– метриками Áльбрехта на основе функциональных указателей.

«Косвенная оценка уровня качества процедурно-ориентированных программных средств с использованием расчетных методов может проводиться на основе применения метрики дефектов качества.

Метрика дефектов качества определяется по формуле 40. Чем больше значение метрики, тем ниже уровень качества программного средства. При этом предполагается соблюдение нормы данной метрики от нуля до единицы.

Указанную метрику качества можно рассчитать не на основе подсчета ошибок, а на основе функциональных указателей Альбрехта» [14]. Áлан Áльбрехт – сотрудник компании IBM [ай би эм], который предложил эту метрику в 1979 году.

В качестве показателя рассматривается не количество строк программы, а функциональные указатели. Несмотря на то, что функциональные указатели определяются субъективно, они при этом характеризуют функциональную сложность программы.

Достоинством такого метода является достаточная легкость вычисления искомых показателей. В то же время результаты оценивания основываются на субъективных данных. Кроме того, используются не прямые, а косвенные измерения.

## МЕТРИКИ НА ОСНОВЕ ФУНКЦИОНАЛЬНЫХ УКАЗАТЕЛЕЙ

Количество функциональных указателей рассчитывают по формуле:

$$FP = F \cdot \left(0{,}65 + 0{,}01 \cdot \sum_{i=1}^{14} k_i\right), \quad (41)$$

где $k_i$ – коэффициенты регулировки сложности;

$F$ – общее количество функциональных указателей:

$$F = \sum_{j=1}^{5} f_j\,, \quad (42)$$

где $f_j$ - оценочные элементы.




Метрики Альбрехта основаны на функциональных указателях. «Количество функциональных указателей является мерой внешних обращений к данным. Под ними понимают ссылки на внешний ввод, внешний вывод, внешний запрос, автономный локальный файл, разделяемый глобальный файл» [14].

Количество функциональных указателей

рассчитывают по формуле 41, которая включает:

- коэффициенты регулировки сложности. Они зависят от ответов на 14 вопросов, связанных с особенностями конкретного программного средства;
- общее количество функциональных указателей, которое определяется по формуле 42 как сумма всех оценочных элементов.

В качестве оценочных элементов используют:

- «количество внешних вводов – ввод данных пользователем – $f_1$ [эф один];
- количество внешних выводов данных – отчеты, экраны, распечатки, сообщения – $f_2$ [эф два];
- количество внешних запросов – диалоговых вводов-выводов – $f_3$ [эф три];
- количество локальных внутренних логических файлов – группа логически связанных данных, которая размещена внутри приложения и

обслуживается через внешние вводы, наборы базы данных – $f_4$ [эф четыре];

– количество внешних интерфейсных файлов – группа логически связанных данных, которая размещена внутри другого приложения и поддерживается им, внутренний логический файл в другом приложении – $f_5$ [эф пять]» [14].

## ОПРЕДЕЛЕНИЕ ЗНАЧЕНИЙ КОЭФФИЦИЕНТОВ РЕГУЛИРОВКИ СЛОЖНОСТИ И ВЕСОВЫХ КОЭФФИЦИЕНТОВ ВАЖНОСТИ

Исходные данные для расчета *FP*-метрик:

| Характеристика | Количество с учетом сложности | | | Итого |
|---|---|---|---|---|
| | Низкая | Средняя | Высокая | |
| Внешние вводы | $m * 3 =$ | $m * 4 =$ | $m * 5 =$ | |
| Внешние выводы | $m * 4 =$ | $m * 5 =$ | $m * 7 =$ | |
| Внешние запросы | $m * 3 =$ | $m * 4 =$ | $m * 6 =$ | |
| Внутренние логические файлы | $m * 7 =$ | $m * 10 =$ | $m * 15 =$ | |
| Внешние интерфейсные файлы | $m * 5 =$ | $m * 7 =$ | $m * 10 =$ | |
| Общее количество | | | | |

С учетом значения *FP* получим новое выражение:

$$DQ = \frac{\text{Количество\_Ошибок}}{FP} = \frac{\text{Количество\_Ошибок}}{\sum_{j=1}^{5} f_j \cdot (0{,}65 + 0{,}01 \cdot \sum_{i=1}^{14} k_i)}. \quad (43)$$

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

Исходные данные для расчета количества функциональных указателей сводятся в таблицу, форма которой приведена на слайде.

В графах таблицы символом *m* [эм] обозначено количество конкретных показателей. При формировании таблицы данный символ заменяется фактическим числом показателей различных характеристик.

«Значения коэффициентов регулировки сложности зависят от ответов на вопросы, касающиеся влияния определенных факторов на выполнение функций программного обеспечения.

Первый вопрос: какое влияние оказывает наличие средств передачи данных?

Второй вопрос: каково влияние распределенной обработки данных?

Третий: каково влияние оказывает распространенность используемой аппаратной платформы?

Четвертый вопрос: какое влияние оказывает критичность к требованиям производительности и ограничению времени ответа?

Пятый вопрос: каково влияние частоты транзакций?

Шестой: каково влияние ввода данных в режиме реального времени?

Седьмой вопрос: какое влияние оказывает эффективность работы конечного пользователя?

Всего приводится 14 вопросов» [14].

Ответы на каждый вопрос должны соответствовать следующим категориям:

– «влияние случайное;
– влияние небольшое, эпизодическое;
– влияние среднее;
– влияние важное, значительное;
– влияние основное, существенное» [14].

Значения весовым коэффициентам важности присваивают от 1 до 5 в зависимости от ответа. Для оценки качества программных средств используют формулу 43.

Наконец, отметим, что областью применения функциональных указателей являются коммерческие информационные системы.

## МЕТРИКИ СВОЙСТВ

Исходные данные для расчета указателя свойств:

| Характеристика | Количество | Сложность | Итого |
|---|---|---|---|
| Вводы | | m*4 = | |
| Выводы | | m*5 = | |
| Запросы | | m*4 = | |
| Логические файлы | | m*7 = | |
| Интерфейсные файлы | | m*7 = | |
| Количество алгоритмов | | m*3 = | |
| Общее количество | | | |

Пересчет *FP*-оценок в *LOC*-оценки:

| Язык программирования | Количество операторов на I FP |
|---|---|
| C++ | 64 |
| Java | 53 |
| Visual C++ | 34 |

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

Для продуктов с высокой алгоритмической сложностью используются метрики свойств оценки качества программных средств. «Они применимы к системному и инженерному программному обеспечению, программному обеспечению реального времени и встроенному ПО» [14].

Для расчета указателя свойств добавляется еще одна характеристика – количество алгоритмов.

Алгоритм в данном случае определяется как ограниченная программа вычислений, включаемая в общую программу.

Исходные данные для расчета указателя свойств представлены в таблице. Можно выделить следующие характеристики:

– вводы;
– выводы;
– запросы;
– логические файлы;
– интерфейсные файлы;
– количество алгоритмов.

После заполнения таблицы значение каждого указателя свойств вычисляется по формуле 41, аналогичной для расчета функциональных указателей *FP* [эф пи].

«Для сложных систем в реальном масштабе времени это значение оказывается на 25-30 % больше

значения, вычисляемого по таблице для количества функциональных указателей.

*FP* [эф пи]-оценки могут быть пересчитаны в *LOC* [лок]-оценки, поскольку известны примерные соотношения значений указателей с количеством строк кода для различных языков программирования» [14].

## МЕТРИКИ СВЯЗНОСТИ МОДУЛЕЙ

Программный модуль представляет собой фрагмент описания процесса обработки данных, оформляемый как самостоятельный программный продукт, который может быть применен для использования в описаниях проектируемого процесса.

Программные модули в рамках создаваемого программного средства обязательно должны быть связаны.

*Связность* представляет собой меру прочности соединения функциональных и информационных объектов в пределах одного программного модуля.

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

«Программные средства являются, как правило, большими системами. Для них следует применять меры упрощения еще на этапе проектирования. Один из применяемых методических приемов – разбиение программы на части, реализуемые в виде программных модулей.

Программные модули физически отделены друг от друга. Кроме того, каждый созданный

программный модуль может включаться в состав разных программ. Это возможно, если выполнены условия его использования, заявленные в документации по этому модулю» [14].

«Программные модули могут рассматриваться, с одной стороны, как способ снижения сложности программ. С другой – как средство, позволяющее бороться с повторением ранее проделанных действий при программировании.

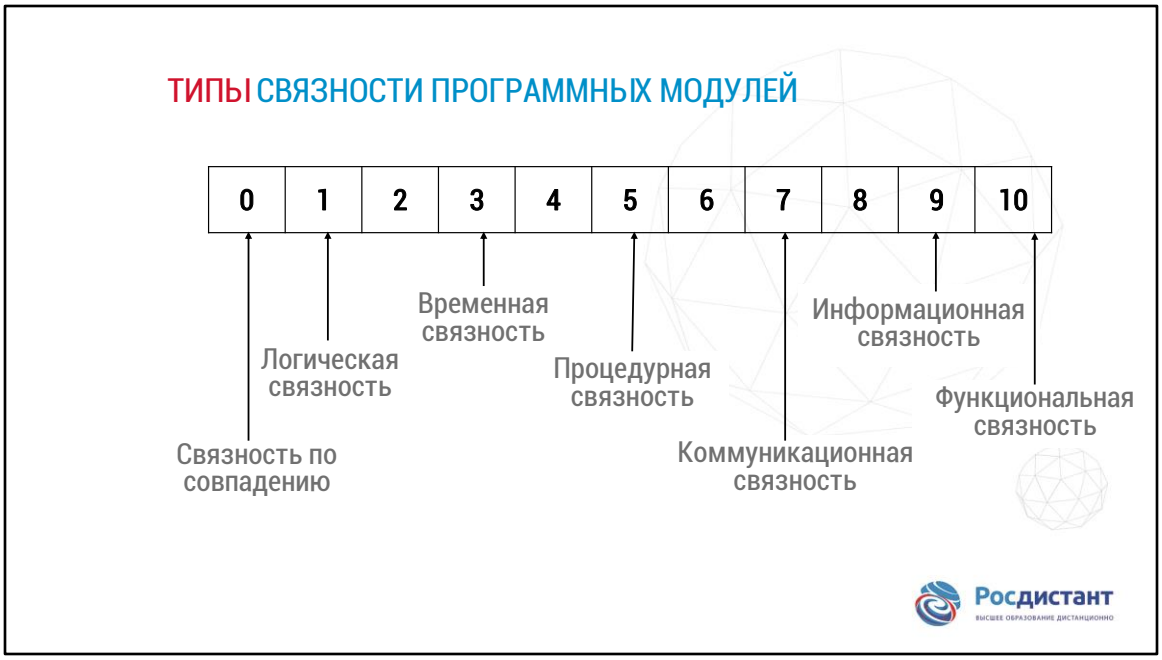
В то же время применение модульной структуры не означает, что каждый модуль должен быть абсолютно автономным. Необходимо обеспечить их согласованное исполнение.

Кроме оценки уровня качества по формуле 40, связывающей количество ошибок программы с размером ее программного кода, можно использовать косвенную оценку уровня качества. В этом случае применяются метрики связности и сцепления

модулей.

Уровень качества программного средства прямо пропорционален силе связности и обратно пропорционален силе сцепления модулей. Для обеспечения необходимого уровня надежности программы разработка качественного программного средства должна предполагать максимизацию связности и минимизацию сцепления модулей» [14].

Размещение сильно связанных объектов в одном модуле существенно уменьшает межмодульные взаимосвязи и их взаимное влияние друг на друга.




Для оценки связности используют целочисленную шкалу силы связности в интервале значений от 0 до 10. С каждым из значений силы связности сопоставлен тип связности, который может быть приписан модулю в процессе проектирования.

«Шкала силы связности использует следующие семь градаций типов связности модулей.

Первый тип – связность по совпадению: сила

связности – 0. Такая связность справедлива для модулей, в которых отсутствуют явные внутренние связи. Случайно связный модуль похож на логически связный: его объекты не связаны ни потоками данных, ни потоками управления.

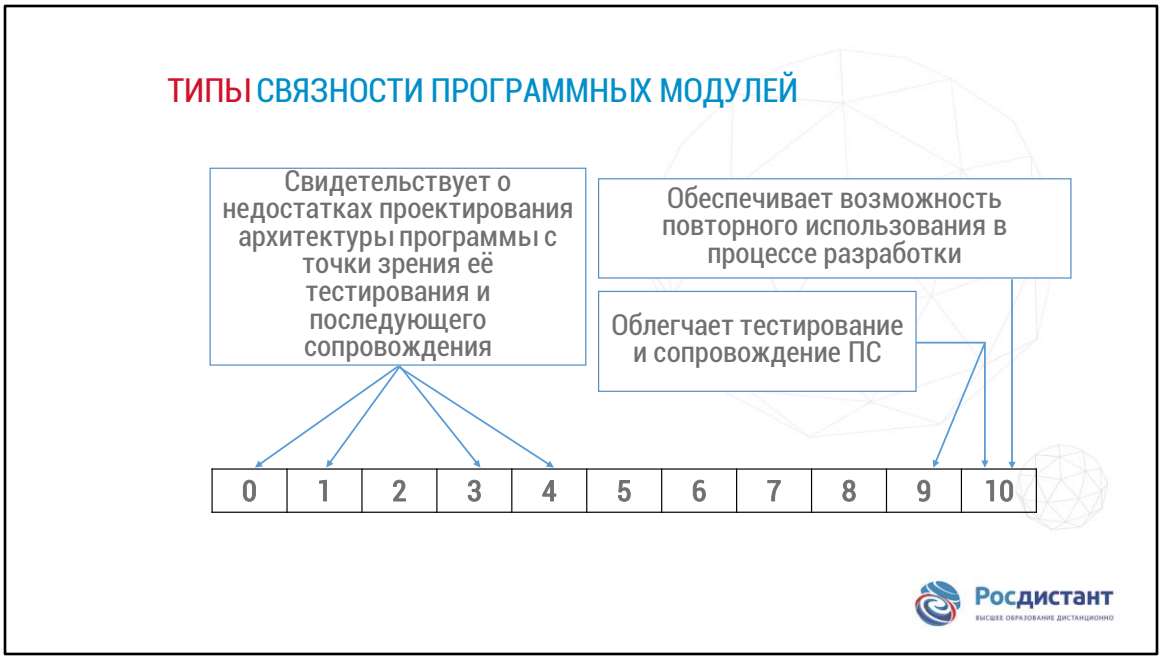
Подзадачи в логически связном модуле относятся к одной категории, а для случайно связанного модуля даже это утверждение является неверным.

Второй тип – логическая связность: сила связности – 1. Такой тип связности характерен для модулей, содержащих подпрограммы для различных вариантов обращения к модулю. Причем каждый раз выполняется одна из подпрограмм модуля.

Это означает, что может быть выполнен любой из возможных вариантов действий. Однако будет запущена только одна подпрограмма, а не все сразу или несколько – последовательно.

Третий тип – временна́я связность: сила связности – 3. Такая связность характерна для модулей, составные части которых не связаны между собой, но должны быть выполнены в течение определенного периода.

Например, в конце рабочего дня перед выключением компьютера предусмотрена проверка рабочих файлов на отсутствие вирусов, копирование на устройства хранения и печать отчета о выполненной работе. Последовательность этих действий не имеет особого значения, однако все эти действия предусматриваются к выполнению в конце рабочего дня» [14].




Продолжим рассматривать градации типов связности модулей.

«Четвертый тип – процедурная связность: сила связности – 5. Данная связность типична для модулей, в которых имеется порядковая связь составляющих их частей, реализующих некоторую предопределенную процедуру обработки данных.

При этом типе связности в цепочке обработки

данных передается именно управление, то есть устанавливается последовательность действий, выполняемых программными модулями.

Пятый тип – коммуникационная связность: сила связности – 7. Такая связность типична для модулей, когда их составляющие используют одну и ту же структуру данных.

Примером является использование данных о книгах, содержащихся в международном стандартном книжном номере: наименование книги, автор, стоимость. Причем разные компоненты этого объекта применяются для решения различных задач одним программным модулем.

Шестой тип – информационная связность: сила связности – 9. Данный тип связности имеет место в случаях, когда выходные данные одной части модуля используются как входные данные другой части модуля. В этом случае объекты охватывают не всю

задачу, а только ее часть – подзадачу.

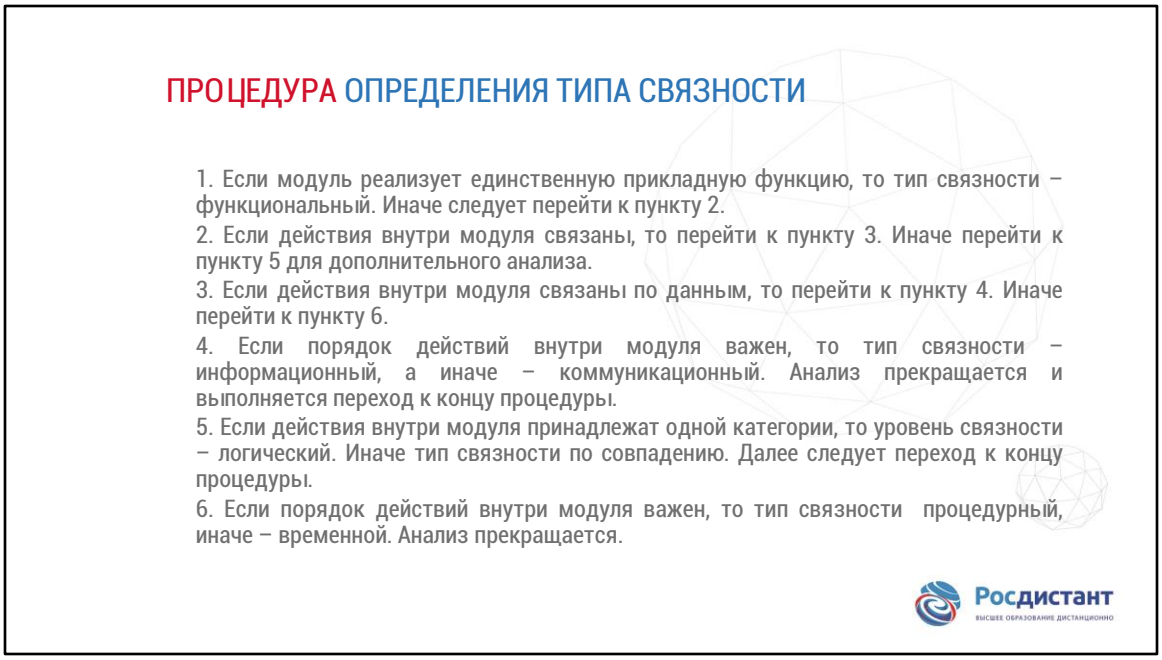
Для такого типа связности выходные данные одной из подзадач служат входными данными для следующей. Примером может служить цепочка выполнения подзадач печати файла: открыть файл, прочитать его и переслать на принтер.

Седьмой тип – функциональная связность: сила связности – 10. Здесь модуль как единое целое реализует одну единственную функцию. Объясняется это тем, что такой программный модуль содержит набор объектов, предназначенных для выполнения только одной задачи» [14].

В качестве примера можно привести расчет показателей успеваемости у конкретной группы. При вызове такого модуля выполняется полностью только одна задача.

Специалисты отмечают, что связность типов первого, второго, третьего или четвертого

свидетельствует о недостатках проектирования архитектуры программы с точки зрения ее тестирования и последующего сопровождения.




Определение типа связности выполняется на основе анализа факторов, характеризующих взаимодействие конкретного программного модуля с остальной частью программного средства, к которому он относится.

«1. Если модуль реализует единственную прикладную проблемно-ориентированную функцию, то тип связности – функциональный. Далее анализ

можно прекратить – конец процедуры. В противном случае следует перейти к пункту 2.

2. Если действия внутри модуля связаны, то нужно перейти к пункту 3. Если же действия внутри модуля ничем не связаны, то следует перейти к пункту 5 для дополнительного анализа.

3. Если действия внутри модуля связаны данными, то перейти к пункту 4, а если связаны потоком управления – к пункту 6.

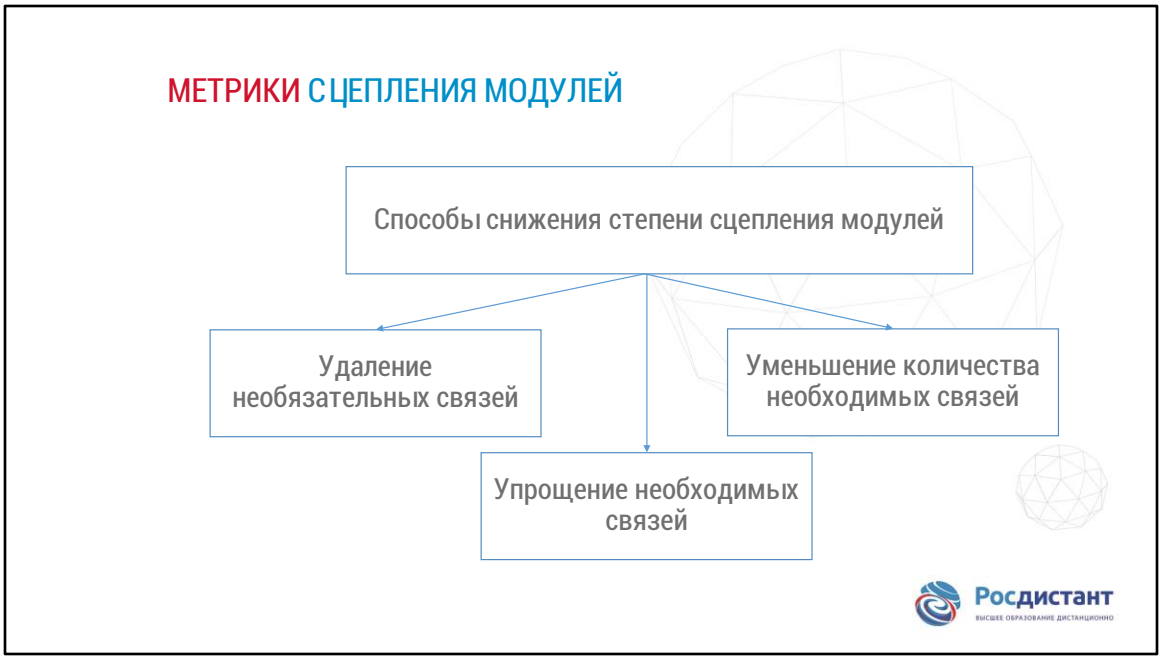
4. Если порядок действий внутри модуля важен, то тип связности – информационный. В противном случае тип связности – коммуникационный. Анализ прекращается и осуществляется переход к концу процедуры.

5. Если порядок действий внутри модуля важен, то тип связности – процедурный. В противном случае тип связности – временно́й. Анализ прекращается.

6. Если действия внутри модуля принадлежат к

одной категории, то уровень связности – логический. Если не принадлежат, значит, модуль обладает типом связности по совпадению. Далее следует переход к концу процедуры» [14].

Несмотря на формализованную методику определения типа связности программного модуля, не всегда удается однозначно определить эту метрику. Если модулю соответствует сразу несколько типов связности, анализируемому программному модулю приписывают тип с наименьшей связностью.




Сцепление определяет меру межмодульной связи, которую для повышения качества программных средств желательно уменьшать. Сцепление позволяет оценить, насколько хорошо программные модули отделены друг от друга.

Слабое сцепление между программными модулями служит характеристикой хорошо спроектированной системы по следующим причинам:

- «уменьшение связи между двумя модулями способствует уменьшению вероятности появления так называемого волнового эффекта. При его проявлении возникновение ошибки в одном модуле неминуемо повлияет на работу других модулей;
- низкая степень связи снижает риск появления так называемого эффекта ряби. Данный эффект проявляется при внесении изменений, так как изменение, как правило, влияет на небольшое количество модулей;
- при сопровождении модуля низкая степень связывания может полностью исключить необходимость заботиться о внутреннем содержании других программных модулей. Иными словами, она позволяет обеспечить своеобразную автономность модулей;
- если модули связаны слабо, то значительно

упрощается контроль их совместного функционирования, а изучение программы существенно упрощает понимание созданной системы» [14].

Следует отметить, что достижение слабого сцепления программных модулей происходит не самопроизвольно. Для этого нужны направленные действия разработчиков.

«Анализ структуры и способов связывания модулей позволяет избавиться от ряда межмодульных связей, которые на деле оказываются необязательными или совсем не нужными.

Искусство программирования к настоящему времени достигло достаточно высокого уровня. Поэтому профессиональное структурирование разрабатываемой программы позволяет достичь снижения количества необходимых связей.

Если связывание конкретных модулей

необходимо, то следует хорошо продумывать наилучшее решение, то есть самое простое.


Для каждой степени сцепления определен тип сцепления, который может быть сопоставлен с модулем в процессе проектирования.

Для измерения сцепления используют целочисленную шкалу степени сцепления в интервале значений от 0 до 9. Шкала степени сцепления использует шесть градаций типов сцепления программных модулей. Типы сцепления

можно охарактеризовать следующим образом.

Первый тип – сцепление по данным: сила сцепления – 1. Модули сцеплены по данным, если они взаимодействуют через передачу параметров и каждый из них является элементарным информационным объектом.

Второй тип – сцепление по образцу: сила сцепления – 3. «При взаимодействии модулей, сцепленных по образцу, вызывающий модуль передает вызываемому составной информационный объект. Например, составной объект «Клиент» содержит много полей.

Третий тип – сцепление по управлению: сила сцепления – 4. Вызывающий модуль передает вызываемому модулю конкретный информационный объект – флаг, который может существенно влиять на изменение внутренней логики модуля. Флаги могут усиливать сцепление модулей и,

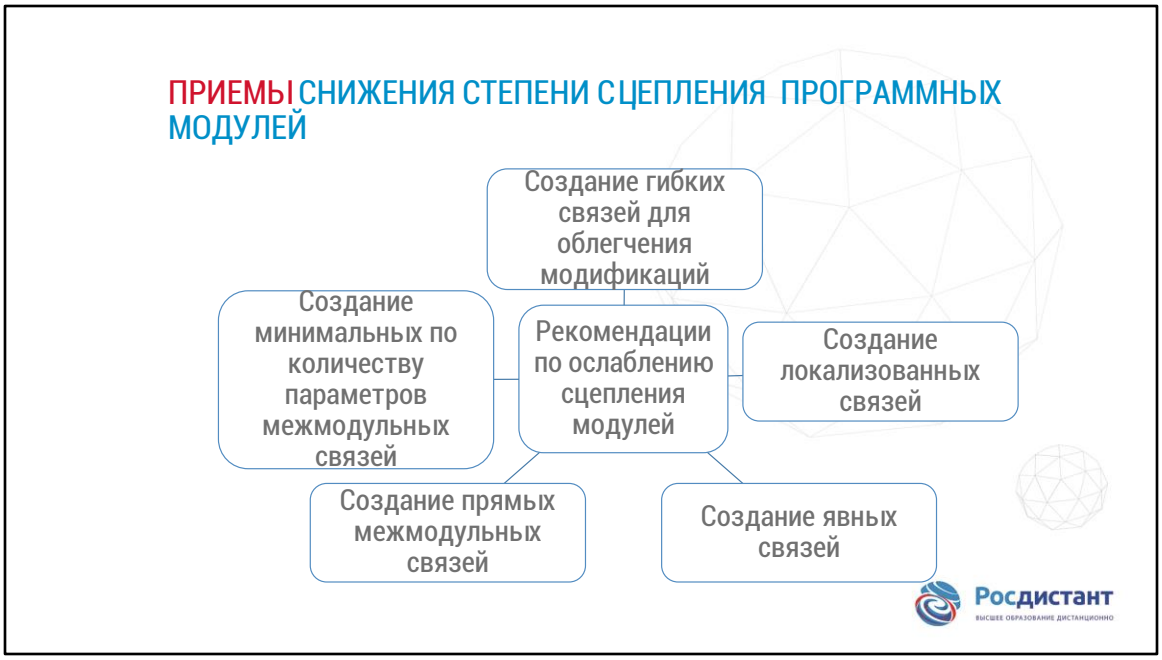
следовательно, ухудшать качество проекта.

Четвертый тип – сцепление по внешним ссылкам: сила сцепления – 5. Такой тип сцепления характерен, например, для блоков констант, определенных в качестве глобальных и взаимодействующих со всеми компонентами анализируемого программного средства. Это довольно плохой тип связывания программных модулей» [14].

Пятый тип – сцепление по общей области: сила сцепления – 7. Модуль использует одну структуру данных совместно с другим модулем. Этот тип связывания, как и предыдущий, также считается плохим.

Шестой тип – сцепление по содержанию: сила сцепления – 9. Программный модуль передает управление или выполняет переход в другой, если один модуль ссылается на значения переменных в

другом модуле или если один модуль изменяет код другого. Этот тип сцепления может ощутимо снизить качество проекта.




К практическим рекомендациям по снижению степени сцепления программных модулей можно отнести применение следующих приемов.

Снижение количества параметров сопряжения программных модулей позволяет практически пропорционально снизить и степень сцепления.

«При непосредственной передаче параметров взаимосвязь между двумя программными модулями

наиболее понятна, поэтому становится проще. Учитывая обычную забывчивость программистов даже в отношении собственных разработок, по прошествии некоторого времени человеку гораздо легче вспомнить и понять такие взаимосвязи без дополнительных обращений к другим информационным объектам.

В данном случае имеет значение даже размещение передаваемых параметров. Так, например, эти значения лучше вычислять непосредственно перед вызовом программного модуля. В этом случае проконтролировать их корректность гораздо легче, чем в случае размещения таких параметров на значительном расстоянии от точки их вызова.

Характерным примером неявной связи является взаимодействие двух программных модулей, построенное на основе модификации области

данных, принадлежащих ко второму модулю. Для того чтобы человек, сопровождающий второй модуль, понял, как модифицируется эта область данных, ему придется проделать немалую дополнительную работу.

Снижение степени сцепления программных модулей или хотя бы гибкость таких связей позволит снизить трудозатраты при любом совершенствовании созданного программного средства.

Жесткое связывание модулей впоследствии потребует бóльших затрат при переделке, чем в случае максимальной автономности компонентов программ» [1].

## ПРИМЕР ОЦЕНКИ ХАРАКТЕРИСТИК ПРОГРАММ НА ОСНОВЕ ПРОЦЕДУРНО-ОРИЕНТИРОВАННЫХ МЕТРИК

Задача «Замена цифр на символы». Фрагмент программы на *C#*:

```
1  using System;
2  class Mymetod
3  {
4  public string change(string sinp,
char sim)
5 {
6  string  buf;
7  int i;
8  for(i=0, buf=""; i<sinp, Length;
i++)
9  if(char.IsDigit(sinp[i]))
10 buf += sim;
11 else
12 buf += sinp[i];
13 return buf;
14 }
15 public static void Main()
16 {
17 string str;
18 char sim, rep;
19 do
20 {
21 Console.WriteLine("Исходная
строка:");
22 str = Console.ReadLine();
...
```

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

Рассмотрим пример оценки характеристик программы на основе процедурно-ориентированных метрик. Необходимо определить класс с методом, выполняющим в строке замену всех цифр символом-заменителем. Строку и символ-заменитель нужно принять через входные параметры.

Требуется также разработать программу и подготовить данные для оценки ее качества на

основе применения функционально-ориентированных метрик. Для этого понадобится определить количество функциональных указателей, уровни связности и сцепления программных модулей.

«Рассмотрим значения функциональных указателей для разработанного решения поставленной задачи:

– количество внешних вводов данных пользователем для рассматриваемой программы составляет 3. Это ввод значений трех переменных в строках 22, 24, 29;
– количество внешних выводов – отчеты, экраны, распечатки, сообщения – составляет 5. Это вывод диалоговых сообщений – строки 21, 23, 26 и 28, вывод результатов на экран – строка 27;
– количество внешних запросов – диалоговых вводов-выводов – равно 0;

– количество локальных внутренних логических файлов равно 0;
– количество внешних интерфейсных файлов также равно 0, так как в решении такие файлы отсутствуют» [14].

Исходные данные для расчета количества функциональных указателей записываются в таблицу с учетом сложности обращений к данным. В соответствии с уровнем сложности значение характеристики умножается на коэффициент сложности.

При внешних вводах решения программы осуществляется обращение к одному элементу данных, что соответствует низкому уровню сложности. При внешних выводах осуществляется обращение к одному элементу данных. Внешние запросы, внутренние и внешние файлы отсутствуют.

## ПРИМЕР ОЦЕНКИ ХАРАКТЕРИСТИК ПРОГРАММЫ НА ОСНОВЕ ПРОЦЕДУРНО-ОРИЕНТИРОВАННЫХ МЕТРИК

Исходные данные для расчета *FP*- метрик:

| Характеристика | Количество с учетом сложности | | | Итого |
|---|---|---|---|---|
| | Низкая | Средняя | Высокая | |
| Внешние вводы | 3 × 3 = 9 | 0 × 4 = 0 | 0 × 5 = 0 | 9 |
| Внешние выводы | 5 × 4 = 20 | 0 × 5 = 0 | 0 × 7 = 0 | 20 |
| Внешние запросы | 0 × 3 = 0 | 0 × 4 = 0 | 0 × 6 = 0 | 0 |
| Внутренние логические файлы | 0 × 7 = 0 | 0 × 10 = 0 | 0 × 15 = 0 | 0 |
| Внешние интерфейсные файлы | 0 × 5 = 0 | 0 × 7 = 0 | 0 × 10 = 0 | 0 |
| | | | Общее количество | 29 |

Количество функциональных указателей вычисляется по формуле (41): $FP = 29 \cdot (0{,}65 + 0{,}01 \cdot 24) = 25{,}81$, где $\sum_{i=1}^{14} k_i = 24$.

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

Значения коэффициентов регулировки сложности определяются ответами на вопросы, которые были приведены ранее. Количество функциональных указателей вычисляется по формуле 41.

На основе рассчитанного значения количества функциональных указателей можно вычислить показатели качества, производительности, удельной стоимости и документированности.

Для определения уровня связности программных модулей полученного решения используется алгоритм процедуры, рассмотренный выше. Представленное решение включает четыре программных модуля.

Метод change [чейндж] осуществляет замену цифр символом-заменителем. Модуль реализует единственную прикладную функцию. Тип связности – функциональный, сила связности равна 10.

Метод Main [мэйн] осуществляет решение поставленной задачи. Модуль реализует не единственную прикладную функцию. Действия внутри модуля связаны. Порядок действия внутри модуля важен. Следовательно, тип связности модуля – коммуникационный, сила связности равна 7.

Таким образом, в решении преобладают модули с силой связности 7 и 10, что говорит о достаточно высоком качестве программы и ее легкой

тестируемости.

Определим уровни сцепления модулей. Для этого проанализируем каждый из модулей разработанного решения на предмет межмодульной связи.

«Метод change [чейндж] является вызываемым, его входные параметры – простые данные. Следовательно, рассматриваемый модуль имеет сцепление по данным. Сила сцепления равна единице.

Метод Main [мэйн] является вызывающим и передает вызываемому модулю списки управляющих параметров, явно влияющих на его работу. Таким образом, метод Main [мэйн] имеет сцепление по управлению. Сила сцепления – 4.

Итак, сила сцепления программных модулей рассматриваемого решения составляет 1 и 4. Это свидетельствует о достаточно высоком уровне

качества программы» [14].