Программная инженерия Лекция 5. Рабочее проектирование

Составитель: Эверстов В.В.

Дата составления: 20/03/2014

Дата модификации: 20/04/2015

Следующий этап какой?

- К этому моменту мы составили ТЗ, обследовали предметную область, разработали архитектуру будущего софта, функциональную и техническую спецификацию. Что делать далее?

Этап рабочего проектирования

- Этап рабочего проектирования это есть само кодирование, разработка БД. В общем, разработка рабочей версии будущей программы.
- В компаниях всегда есть стандарты (правила) оформления исходного кода. Эти правила направлены на повышение читабельности кода и легкости его сопровождения.

Оформление исходного кода

- Что включают правила оформления кода?
  - Правила выбора названий (переменных, методов, классов)
  - Стиль отступов при оформлении логических блоков,
  - Способы расстановки блоков,
  - Использование пробелов при оформлении логических и арифметических выражений
  - Оформление документирующих комментариев

Правила наименований

- При наименовании переменных, методов, классов существуют множество правил, вы даже можете придумать свои. Вот примеры таких правил:
  - «Верблюжий стиль»
  - Венгерская нотация.

Верблюжий стиль

- ***Верблюжий стиль – Верблюжий Регистр — Горбатый Стиль***.
- Стиль написания составных слов, при котором несколько слов пишутся слитно без пробелов, при этом каждое слово пишется с заглавной буквы. Стиль получил название *CamelCase*, поскольку заглавные буквы внутри слова напоминают горбы *верблюда*
- Для названия классов используют **Upper**CamelCase, для методов и объектов класса — **lower**CamelCase.

Венгерская нотация

- Соглашение об именовании переменных, констант и прочих идентификаторов в коде программ. Своё название венгерская нотация получила благодаря программисту компании Майкрософт венгерского происхождения Чарльзу Симони. Эта система стала внутренним стандартом Майкрософт.
- Суть венгерской нотации сводится к тому, что имена идентификаторов предваряются заранее оговорёнными префиксами, состоящими из одного или нескольких символов. При этом, как правило, ни само наличие префиксов.

Венгерская нотация

Префикс	Сокращение от	Смысл	Пример
s	string	строка	sClientName
sz	zero-terminated string	строка, ограниченная нулевым символом	szClientName
n, i	int	целочисленная переменная	nSize, iSize
l	long	длинное целое	lAmount
b	boolean	булева переменная	bIsEmpty
a	array	Массив	aDimensions
t, dt	time, datetime	время, дата и время	tDelivery, dtDelivery
p	pointer	Указатель	pBox
lp	long pointer	двойной (дальний) указатель	lpBox
r	reference	Ссылка	rBoxes
h	handle	Дескриптор	hWindow
m_	member	переменная-член	m_sAddress
g_	global	глобальная переменная	g_nSpeed
C	class	Класс	CString
T	type	Тип	TObject
I	interface	Интерфейс	IDispatch
v	void	отсутствие типа	vReserved

Оформление логических блоков

- Отступы в языке C:
  - Стиль K&R
  - Стиль Олмана
  - Стиль Уайтсмита
  - Стиль GNU

Стиль K&R

- Назван в честь Кернигана и Ричи из-за того, что все примеры из их книги «Язык программирования Си» отформатированы подобным образом.

```
if (<cond>) {
........<body>
}
```

- Основной отступ состоит из 8 пробелов (или одной табуляции) на уровень. Чаще всего используется 4 пробела.

Стиль Олмана

- *Стиль Олмана* — по имени Эрика Олмана, хакера из Беркли, написавшего множество BSD-утилит на нём (еще известен как «стиль BSD»).

```
if (<cond>)
{
........<body>
}
```

- Основной отступ на уровень — 8 пробелов, но не менее распространен стиль в 4 пробела (особенно в C++).

Стиль Уайсмита

- популярен из-за примеров, шедших с Whitesmiths C — ранним компилятором с языка C. Основной отступ на уровень для скобок и блока — 8 пробелов.

```
if (<cond>)
........{
........<body>
........}
```

Стиль GNU

- *Стиль GNU* — используется во всех исходниках проекта GNU (например, GNU Emacs). Отступы всегда 4 символа на уровень, скобки находятся на половине отступа.

```
if (<cond>)
··{
····<body>
··}
```

Использование пробелов

- Это правило гласит, что любой оператор должен быть отделен от переменных и скобок пробелом:
- a := b + c * (d + e)

Комментарии

- Комментарии повышают читабельность кода.
- Лучше код комментировать, т.к. ваш код могут прочитать другие программисты. Через какое-либо время вы можете позабыть для чего предназначен тот или иной код.
- Каждый метод предваряют комментариями:
  - Автор,
  - Дата создания,
  - Краткое описание,
  - Аргументы,
  - Возвращаемый тип.

Какой стиль программирования выбрать?

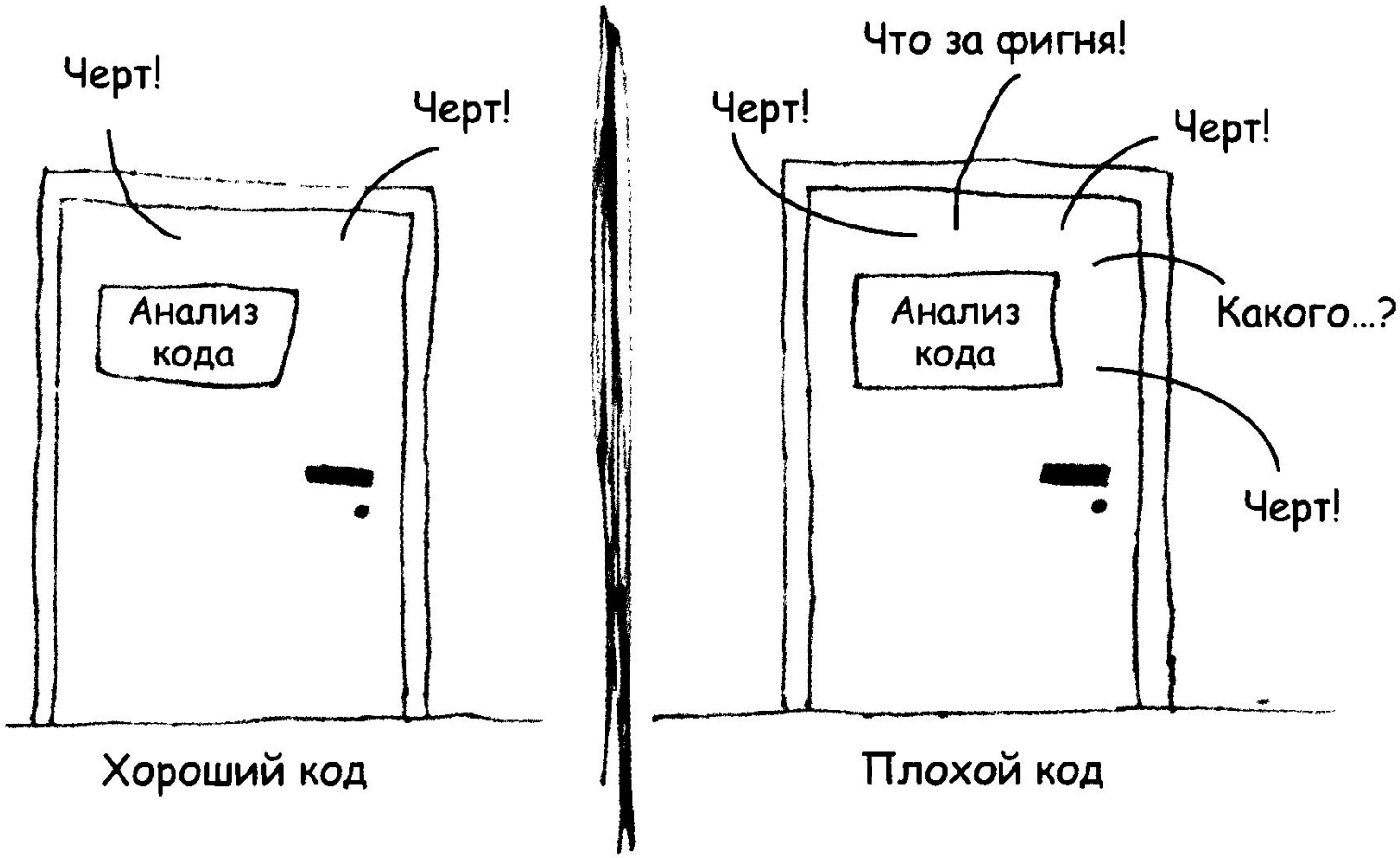
- Выбор того или иного стиля программирования зависит не только от ваших предпочтений но в большей степени и от языка программирования, которое вы выбрали для разработки программного обеспечения.

Чистый код

- Как определить какой код лучше и какой код является ли чистым, хорошим? Есть ли какие-либо критерии выявления чистого кода?

Критерии

Единственная надежная метрика качества кода:
количество «чертей» в минуту



Разработка классов

- Первый и, наверное, самый важный этап разработки высококачественного класса — создание адекватного интерфейса. Это подразумевает, что интерфейс должен представлять хорошую абстракцию, скрывающую детали реализации класса.

Интерфейс

- Выражайте в интерфейсе класса согласованный уровень абстракции

```
class EmployeeCensus: public ListContainer {
public:
// открытые методы
```
Абстракция, формируемая этими методами, относится к уровню «employee» (сотрудник).
```
void AddEmployee( Employee employee );
void RemoveEmployee( Employee employee );
```
Абстракция, формируемая этими методами, относится к уровню «list» (список).
```
Employee NextItemList();
Employee FirstItem();
Employee LastItem();
    …
private:
    …
```

Интерфейс

- Согласованный интерфейс

```
class EmployeeCensus {
public:
// открытые методы
```
Абстракция, формируемая всеми этими методами, теперь относится к уровню «employee».
```
void AddEmployee( Employee employee );
void RemoveEmployee( Employee employee );
Employee NextEmployeeO;
Employee FirstEmployee();
Employee LastEmployeeO;
private:
```
Тот факт, что класс использует библиотеку ListContainer, теперь скрыт.
```
ListContainer m_EmployeeList;
};
```

Интерфейс

- Убедитесь, что вы понимаете, реализацией какой абстракции является класс.
- Предоставляйте методы вместе с противоположными им методами
- Убирайте постороннюю информацию в другие классы
- Опасайтесь нарушения целостности интерфейса при модификации и расширении.

```
class Employee {
public:
   ...
   // открытые методы
   FullName GetName() const;
   Address GetAddress() const;
   PhoneNumber GetWorkPhone() const;
   ...
   bool IsJobClassificationValid( JobClassification jobClass );
   bool IsZipCodeValid( Address address );
   bool IsPhoneNumberValid( PhoneNumber phoneNumber );

   SqlQuery GetQueryToCreateNewEmployee() const;
   SqlQuery GetQueryToModifyEmployee() const;
   SqlQuery GetQueryToRetrieveEmployee() const;
   ...
private:
   ...
};
```

Интерфейс

- Не включайте в класс открытые члены, плохо согласующиеся абстракцией интерфейса.
- Рассматривайте абстракцию и связность вместе

Инкапсуляция

- Минимизируйте доступность классов и их членов
- Не делайте данные-члены открытыми
- Не включайте в интерфейс класса закрытые детали реализации
- Не делайте предположений о клиентах класса
- Избегайте использования дружественных классов
- Не делайте метод открытым лишь потому, что он использует только открытые методы
- Цените легкость чтения кода выше, чем удобство его написания
- Очень, очень настороженно относитесь к семантическим нарушениям инкапсуляции.

Не следует

- Избегайте создания «божественных» классов,
- Устраняйте нерелевантные классы,
- Избегайте классов, имена которых напоминают глаголы.

Методы

```
void HandleStuff( CORP_DATA & inputRec, int crntQtr, EMP_DATA empRec,
   double & estimRevenue, double ytdRevenue, int screenX, int screenY,
   COLOR_TYPE & newColor, COLOR_TYPE & prevColor, StatusType & status,
   int expenseType )
{
int i;
for ( i = 0; i < 100; i++ ) {
   inputRec.revenue[i] = 0;
   inputRec.expense[i] = corpExpense[ crntQtr ][ i ];
   }
UpdateCorpDatabase( empRec );
estimRevenue = ytdRevenue * 4.0 / (double) crntQtr;
newColor = prevColor;
status = SUCCESS;
if ( expenseType == 1 ) {
     for ( i = 0; i < 12; i++ )
           profit[i] = revenue[i] - expense.type1[i];
     }
else if ( expenseType == 2 )  {
          profit[i] = revenue[i] - expense.type2[i];
          }
else if ( expenseType == 3 )
          profit[i] = revenue[i] - expense.type3[i];
          }
```

Имена методов

- Описывайте все, что метод выполняет
  - Опишите в имени метода все выходные данные и все побочные эффекты. Если метод вычисляет сумму показателей в отчете и открывает выходной файл, имя ComputeReportTotals() не будет адекватным. ComputeReportTotalsAndOpenOutputFile() — имя адекватное, но слишком длинное и несуразное.
- Избегайте невыразительных и неоднозначных глаголов
  - могут описывать практически любое действие. Имена вроде HandleCalculation(), PerformServices(), OutputUser(), ProcessInput() и DeatWithOutput() не говорят о работе методов почти ничего. В лучшем случае по этим именам можно догадаться, что методы имеют какое-то отношение к вычислениям, сервисам, пользователям, вводу и выводу соответственно.

Имена методов

- Не используйте для дифференциации имен методов исключительно номера
  - Один разработчик написал весь свой код в форме единственного объемного метода. Затем он разбил код на фрагменты по 15 строк и создал методы Part1, Part2 и т. д.
- Не ограничивайте длину имен методов искусственными правилами
  - Исследования показывают, что оптимальная длина имени переменной равняется в среднем 9-15 символам. Как правило, методы сложнее переменных, поэтому и адекватные имена методов обычно длиннее.

Имена методов

- Для именования функции используйте описание возвращаемого значения
  - Функция возвращает значение, и это следует должным образом отразить в ее имени. Так, имена cos(), customerId.Next(), printer.IsReady() и pen.CurrentColor() ясно указывают, что возвращают функции, и потому являются удачными.
- Для именования процедуры используйте выразительный глагол, дополняя его объектом
  - его объектом Процедура с функциональной связностью обычно выполняет операцию над объектом. Имя должно отражать выполняемое процедурой действие и объект, над которым оно выполняется, что приводит нас к формату «глагол + объект». Примеры удачных имен процедур — PrintDocument(), CalcMontblyRevenues(), CheckOrderInfo() и RepaginateDocument().
  - В случае объектно-ориентированных языков имя объекта в имя процедуры включать не нужно, потому что объекты и так входят в состав вызовов, принимающих вид document.Print(), orderInfo.Check() и monthlyRevenues.Calc(). Имена вида document.PrintDocument() страдают от избыточности и могут стать в производных классах неверными.

Имена методов

- Дисциплинированно используйте антонимы
  - Применение конвенций именования, подразумевающих использование антонимов, поддерживает согласованность имен, что облегчает чтение кода. Антонимы вроде first/last понятны всем. Пары вроде FileOpen() и _lclose() несимметричны и вызывают замешательство.
- Определяйте конвенции именования часто используемых операций
  - При работе над некоторыми системами важно различать разные виды операций. Самым легким и надежным способом определения этих различий часто оказывается конвенция именования.
    - employee.id.Get()
    - dependent.GetId()
    - supervisor()
    - candidate.id()

Параметры

- Передавайте параметры в порядке «входные значения – изменяемые значения — выходные значения»
- Подумайте о создании собственных ключевых слов in и out
- Используйте все параметры
- Передавайте переменные статуса или кода ошибки последними
- Не используйте параметры метода в качестве рабочих переменных
- Документируйте выраженные в интерфейсе предположения о параметрах
- Ограничивайте число параметров метода примерно семью 7

Возвращаемые значения

- Проверяйте все возможные пути возврата.
  - Создав функцию, проработайте в уме каждый возможный путь ее выполнения, дабы убедиться, что функция возвращает значение во всех возможных обстоятельствах. Целесообразно инициализировать возвращаемое значение в начале функции значением по умолчанию: это будет страховкой на тот случай, если функция не установит корректное возвращаемое значение.
- Не возвращайте ссылки или указатели на локальные данные
  - Как только выполнение метода завершается и локальные данные выходят из области видимости, ссылки и указатели на локальные данные становятся некорректными. Если объект должен возвращать информацию о своих внутренних данных, пусть он сохранит ее в форме данных — членов класса. Реализуйте для него функции доступа, возвращающие данные-члены, а не ссылки или указатели на локальные данные.

Case средства

- Термин CASE (Computer Aided Software Engineering) используется в настоящее время в весьма широком смысле.
- Первоначальное значение термина CASE, ограниченное вопросами автоматизации разработки только лишь программного обеспечения (ПО), в настоящее время приобрело новый смысл, охватывающий процесс разработки сложных ИС в целом.

Case средства

- Обычно к CASE-средствам относят любое программное средство, автоматизирующее ту или иную совокупность процессов жизненного цикла ПО и обладающее следующими основными характерными особенностями:
  - мощные графические средства для описания и документирования ИС, обеспечивающие удобный интерфейс с разработчиком и развивающие его творческие возможности;
  - интеграция отдельных компонент CASE-средств, обеспечивающая управляемость процессом разработки ИС;
  - использование специальным образом организованного хранилища проектных метаданных (репозитория).

Case средства

- Интегрированное CASE-средство (или комплекс средств, поддерживающих полный ЖЦ ПО) содержит следующие компоненты;
  - **репозиторий**, являющийся основой CASE-средства. Он должен обеспечивать хранение версий проекта и его отдельных компонентов, синхронизацию поступления информации от различных разработчиков при групповой разработке, контроль метаданных на полноту и непротиворечивость;
  - **графические средства анализа и проектирования**, обеспечивающие создание и редактирование иерархически связанных диаграмм (DFD, ERD и др.), образующих модели ИС;
  - **средства разработки приложений**, включая языки 4GL и генераторы кодов;
  - **средства конфигурационного управления;**
  - **средства документирования;**
  - **средства тестирования;**
  - **средства управления проектом;**
  - **средства реинжиниринга.**

Классификация CASE-средств

- средства анализа (Upper CASE), предназначенные для построения и анализа моделей предметной области (Design/IDEF (Meta Software), BPwin (Logic Works));
- средства анализа и проектирования (Middle CASE), поддерживающие наиболее распространенные методологии проектирования и использующиеся для создания проектных спецификаций (Vantage Team Builder (Cayenne), Designer/2000 (ORACLE), Silverrun (CSA), PRO-IV (McDonnell Douglas), CASE.Аналитик. Выходом таких средств являются спецификации компонентов и интерфейсов системы, архитектуры системы, алгоритмов и структур данных;
- средства проектирования баз данных, обеспечивающие моделирование данных и генерацию схем баз данных (как правило, на языке SQL) для наиболее распространенных СУБД. К ним относятся ERwin (Logic Works), S-Designor (SDP) и DataBase Designer (ORACLE).
- средства разработки приложений. К ним относятся средства 4GL (Uniface (Compuware), JAM (JYACC), PowerBuilder (Sybase), Developer/2000 (ORACLE), New Era (Informix), SQL Windows (Gupta), Delphi (Borland) и др.);
- средства реинжиниринга, обеспечивающие анализ программных кодов и схем баз данных и формирование на их основе различных моделей и проектных спецификаций. Средства анализа схем БД и формирования ERD входят в состав Vantage Team Builder, PRO-IV, Silverrun, Designer/2000, ERwin и S-Designor.

Вспомогательные типы

- Средства планирования и управления проектом (SE companion, microsoft project и др.);
- Средства конфигурационного управления (pvcs (intersolv));
- Средства тестирования (quality works (segue software));
- Средства документирования (soda (rational software)).

Генерация документов

- Для создания документации в процессе разработки используются разнообразные средства формирования отчетов, а также компоненты издательских систем.

Средства тестирования

- Одно из наиболее развитых средств тестирования QA (новое название - Quality Works) представляет собой интегрированную, многоплатформенную среду для разработки автоматизированных тестов любого уровня, включая тесты регрессии для приложений с графическим интерфейсом пользователя.