

Объектно-ориентированный Анализ и Дизайн

Часть 2
 OO Анализ: Аналитическая модель
 Переход от анализа к дизайну
 Принципы OO дизайна

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 1

5. OO Анализ

- Цели анализа
- Аналитическая модель
- Аналитические классы и отношения
- Реализация use-cases
- Диаграммы деятельности и состояний
- Диаграммы взаимодействия
- Трансформация анализа в дизайн

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 2

Цели фаз анализа и дизайна

Задачи:

- Трансформировать требования собранные на предыдущем этапе в дизайн системы
- Проработать архитектуру системы
- Адаптировать дизайн к среде исполнения

Модели:

- Аналитическая модель (Analysis model)
- Design model

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 3

Аналитическая модель

➤ Абстрактная модель системы, описывающая ее в терминах use-case realization. Язык реализации классов не фиксируется. Обычно не сопровождается.

➤ Элементы analysis model:

- Use-case realization** – реализация use-case, набор activity, state, collaboration и class диаграмм
- Boundary class** – класс, разграничивающий actor-ов и систему
- Control** – класс, управляющий другими классами
- Entity** – класс, моделирующий информацию, используемую в системе

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 4

Boundary class

-Класс, разграничивающий (под-)систему и окружение.

UML: class со стереотипом <<boundary>>

Примеры: классы пользовательского интерфейса, классы интерфейсов систем и устройств

<<boundary>> A.PIN code entry form
- pin : String
+ OK() : void
+ Cancel() : void

A.PIN code entry form

Представление boundary посредством стереотипа и пиктограммы

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 5

Control

-Класс, управляющий другими классами. Можно сказать, что control "исполняет" use-case

UML: class со стереотипом <<control>>

<<control>> A.Authorization controller
+ validate(code : A.PIN code) : boolean
- getCCInfo() : A.CCInfo

A.Authorization controller

Представление control посредством стереотипа и пиктограммы

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 6

Entity

-Класс, моделирующий информацию, используемую в системе

UML: class со стереотипом <<entity>>

<<entity>> A.CCInfo
+ <<stereotype>> number : String
+ owner : String

A.CCInfo

Представление entity посредством стереотипа и пиктограммы

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 7

Диаграммы взаимодействия

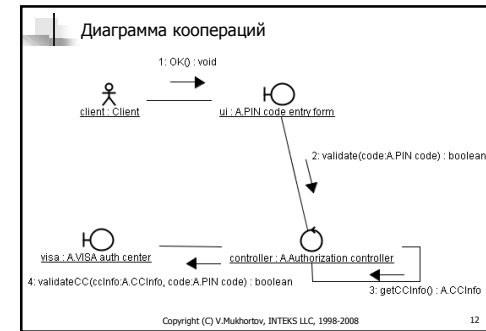
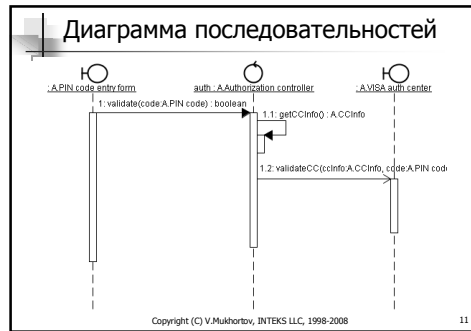
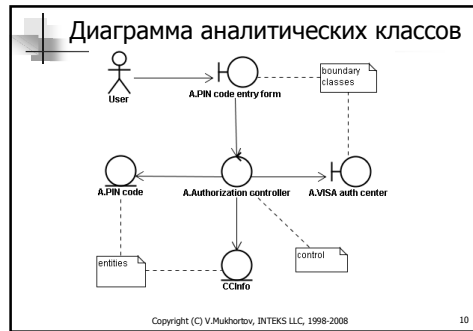
- Последовательностей - Sequence diagrams
- Коопераций - Collaboration diagrams

- Отражают динамические аспекты поведения объектов
- Семантически эквивалентны
- Содержат:
 - Объекты
 - Связи
 - Сообщения
 - Потоки данных

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 8

Авторизация в банкомате

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 9



Ограничения на связи

From/To (navigability)	Boundary	Entity	Control
Boundary	yes	yes	yes
Entity	no*	yes	no*
Control	yes	yes	yes

* Используйте обратные связи со стереотипом "subscribe-to"

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 13

6. OO-дизайн

- Дизайн классов
- Дизайн пакетов
- Поиск и применение шаблонов

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 14

- ### Цели дизайна
- Адаптировать аналитическую модель к конкретным языкам и технологиям, выбранным для реализации системы, т.е:
 - Завершить проработку архитектуры системы (выбор платформ, технологий, библиотек, компонентов, протоколов...)
 - Проработать дизайн (слои, пакеты, межпакетные интерфейсы, ключевые абстракции)
 - При этом обеспечить:
 - Соответствие нефункциональным требованиям
 - Тестопригодность
 - Расширяемость системы (extensibility)
 - Легкость поддержки (maintainability)
 - Создание переиспользуемых компонент (reusability)
- Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 15

Design model

- Модель реализации системы. Создается на основе аналитической модели. Фиксирует язык реализации классов и используемые API. Сопровождается до конца разработки.
- Элементы design model:
 - **Layer** - слой (UI, UI logic, Business logic, Data, system)
 - **Subsystem** - подсистема
 - **Package** - пакет
 - **Class** - класс
 - **Use-case realization** - коллекция диаграмм

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 16

Переход от анализа к дизайну

- Аналитический класс при переходе к дизайну трансформируется в один или несколько классов дизайна, которые реализуются на каком-либо конкретном языке программирования

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 17

Трансформация boundary

- Классы пользовательского интерфейса

- Сколько объектов скольких классов вы можете найти на форме ввода PIN кода справа?

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 18

Трансформация boundary

- Интерфейс (для) внешней системы

Аналитическая модель	Дизайн модель
<pre> classDiagram class A.MSA_auth_center { <<boundary>> +validate(CcInfo, CcInfo, code : APIN code) : boolean } </pre>	<pre> classDiagram class VisaAuthority { <<boundary>> +validate(CcInfo, CcInfo, pinCode : String) : boolean } </pre>

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 19

Трансформация entity

- Класс(ы) типов данных, специфичных для предметной области

Аналитическая модель	Дизайн модель
<pre> classDiagram class A.CCInfo { } </pre>	<pre> classDiagram class VisaCCInfo { - ccNumber : long - owner : String - bank : String + CCInfo(number : int, owner : String, bank : String) : void } </pre>

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 20

Трансформация control

- Один или несколько интерфейсов, реализованные пакетом или группой пакетов (подсистема).

```

classDiagram
    class VisaAuthController {
        + validate(code : String) : boolean
        - getCCInfo() : VisaCCInfo
    }
    class VisaAuthorityAgent {
        + validate(CcInfo : VisaCCInfo, code : String) : boolean
    }
    class A.Authorization_controller {
        <<control>>
        + validate(code : APIN code) : boolean
        - getCCInfo() : A.CCInfo
    }
    class VisaCCInfo {
        - ccNumber : String
        - owner : String
        - bank : String
        + CCInfo(number : String, owner : String, bank : String) : void
    }
    VisaAuthController ..|> VisaAuthorityAgent
    A.Authorization_controller ..|> VisaAuthController
    A.Authorization_controller ..|> VisaCCInfo
        
```

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 21

7. Принципы ОО дизайна

- Вопрос о том, как пишут хорошие программы на C++, похож на вопрос о том, как пишут хорошую английскую прозу.

Б.Страуструп

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 22

Принципы дизайна классов

- ORR** - Принцип целостности абстракции (One Responsibility Rule)
- LSP** - Принцип подстановки (Liskov Substitution Principle)
- LoD** - Закон Деметры (Law of Demeter)
- OCP** - Принцип закрытости абстракции (Open-Closed Principle)
- ISP** - Разделение интерфейсов (Interface Segregation Principle)

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 23

ORR – Правило целостности абстракции

Кот с улыбкой – и то редкость, но уж улыбка без Кота – это я прямо не знаю что такое! Алиса в стране чудес

Класс должен обладать единственной ответственностью,

- ✓ реализует ее полностью,
- ✓ реализует ее хорошо,
- ✓ реализует только ее

A class has a single responsibility:

- ✓ it does it all,
- ✓ it does it well,
- ✓ it does it only

- R. Martin

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 24

LSP – Принцип подстановки

- ✓ Поведение методов, принимающих в качестве параметра указатели и ссылки на объекты базового класса, не должно зависеть от того, к какому классу (базовому или любому из производных) принадлежит переданный объект. - R.Martin, 1996
- ✓ Оригинальная формулировка:
If for each object o1 of type S there is an object o2 of type T such that for all programs P defined in terms of T the behavior of P is unchanged when o1 is substituted by o2 then S is a subtype of T.

- Barbara Liskov, 1988

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 25

Нарушение LSP: Фигуры

```

classDiagram
    class Rectangle {
        private int h;
        private int w;
        public Rectangle(int w, int h) { this.h = h; this.w = w; }
        public void setHeight(int h) { this.h = h; }
        public int getHeight() { return h; }
    }
    class Square extends Rectangle {
        private int s;
        public Square(int s) { super(s, s); }
    }
        
```

Проблема:
Square s = new Square(5);
s.setHeight(6); // Объект s перестал быть квадратом

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 26

Пробуем исправить дело:

```

classDiagram
    class Rectangle {
        private int h;
        private int w;
        public Rectangle(int w, int h) { this.h = h; this.w = w; }
        public void setHeight(int h) { this.h = h; }
        public int getHeight() { return h; }
    }
    class Square extends Rectangle {
        private int s;
        public Square(int s) { super(s, s); }
        public void setSize(int s) { super.setSize(s); super.setHeight(s); }
        public void setHeight(int h) { setSize(h); }
        public void setWidth(int w) { setSize(w); }
    }
        
```

Не помогает:
void f(Rectangle r) throws Exception {
r.setHeight(4);
r.setWidth(5);
if(r.getHeight() * r.getWidth() != 20) throw new Exception("Bug!");
}

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 27

LSP: в чем проблема?

- С точки зрения ОО подхода квадрат просто **не является** прямоугольником, потому что:
- Класс – абстракция данных *и поведения*
- При этом *поведение* квадрата **существенно отличается** от поведения прямоугольника
- Наследование – отношение «частное-общее», но общность надо искать в поведении, а не в структуре

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 28

LoD – Закон Деметры

- Метод должен обладать ограниченным знанием об объектной модели приложения. - D. Rumbaugh
- Оригинальная формулировка:
Only talk to your immediate friends. Never talk to strangers. - Ian Holland, 1987

Друзья метода **f**:

- методы класса **f** и классы параметров метода **f**
- методы классов - полей класса **f**
- методы классов объектов, создаваемых в **f**.

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 29

Нарушение LoD

```

classDiagram
    class MyDialog {
        +getButton() Button
        +show() void
    }
    class Button {
        +disable() void
    }
    class Application {
        +showMyDialog() void
    }
    MyDialog --> Button
    Application --> MyDialog
    
```

Проблема: public void showMyDialog() {
... // создание MyDialog
myDialog.getButton().disable();
}

- связь Application -> Button мы не планировали.
- Замена класса Button на другой интерфейсный класс ведет к перекомпиляции и перестроиванию Application

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 30

LoD-совместимый дизайн

```

classDiagram
    class MyDialog {
        +getButton() Button
        +show() void
        +disableOkButton() void
    }
    class Button {
        +disable() void
    }
    class Application {
        +showMyDialog() void
    }
    MyDialog --> Button
    Application --> MyDialog
    
```

Решение: void showMyDialog() {
... // создание MyDialog
Dialog.disableOkButton(); // секрет класса MyDialog останется секретом
}

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 31

ОСР – Принцип закрытости абстракции

- Компоненты программной системы (классы, модули, методы) должны быть открыты для расширения, но закрыты от модификации. - B. Meyer, 1988
- Не давая каких-либо общих рецептов, этот принцип заставляет нас думать о возможных *изменениях* кода под воздействием изменяющихся требований
- Смысл здесь в том, что добавление новой функциональности должно скорее приводить к **добавлению** нового кода, нежели к модификации существующего. В идеале – только к написанию новых классов.

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 32

Нарушение ОСР: Фигуры

```

classDiagram
    class Shape {
        +SQUARE : int = 1
        +CIRCLE : int = 2
        +getType()
    }
    class Circle {
        +getType()
    }
    class Square {
        +getType()
    }
    Shape <|-- Circle
    Shape <|-- Square
    
```

```

void drawShapes( Shape[] shapes )
{
    for( int i = 0; i < shapes.length; ++i )
        if( shape[i].getType == Shape.SQUARE )
            drawSquare( (Square)shape[i] );
        else drawCircle( (Circle)shape[i] );
}
    
```

Проблема:
Нельзя добавить в систему новый тип фигур, не изменив класса Shape и метода drawShapes().

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 33

ОСР-совместимое решение

```

classDiagram
    class Shape {
        <<Interface>>
        +draw(device : DrawDevice) void
    }
    class DrawDevice {
        <<Interface>>
    }
    class Circle {
        +draw()
    }
    class Square {
        +draw()
    }
    class Triangle {
        +draw()
    }
    Shape <|-- Circle
    Shape <|-- Square
    Shape <|-- Triangle
    
```

```

void drawShapes( Shape[] shapes ) {
    for( int i=0; i < shapes.length; ++i )
    {
        shape[i].draw( device );
    }
}
    
```

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 34

ISP – Разделение интерфейсов

- Клиентов нельзя заставлять платить за сервисы, которых они не используют. - R.Martin, 1996

Hints:

- Избегайте «толстых» интерфейсов
- Разные клиенты – разные интерфейсы

Цена нарушения:

- Потеря гибкости

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 35

Нарушение ISP: Security Door

```

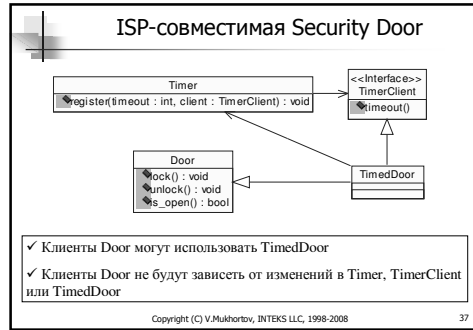
classDiagram
    class Timer {
        +register(timeout : int, client : TimerClient) void
    }
    class Door {
        <<Interface>>
        +lock() void
        +unlock() void
        +open() bool
    }
    class TimerClient {
        +lock() void
        +unlock() void
        +open() bool
    }
    Door <|-- TimerClient
    
```

Дверь издает звук если открыта слишком долго.

Проблемы:

- Метод *timeout()* обязан быть public.
- Некоторые клиенты класса Door не используют и не должны использовать *timeout()*.
- Может приводить к ошибкам.

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 36



Дизайн связей

- **DIP** Dependency Inversion Principle (инверсия зависимостей)
- **ADP** Acyclic Dependencies Principle (ацикличность зависимостей)

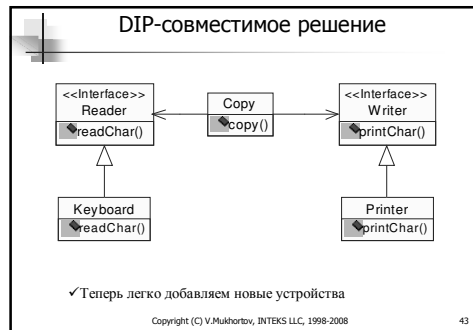
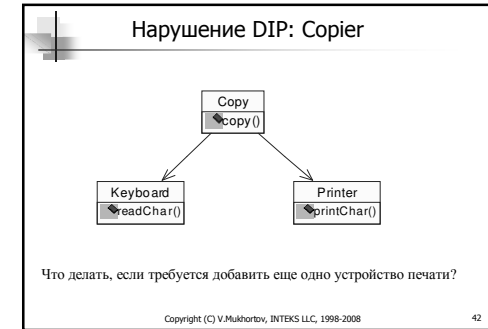
Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 40

DIP – Инверсия зависимостей

- ✓ Модули «высокого» уровня не должны зависеть от модулей «низкого» уровня. И те и другие должны зависеть от абстракций.
- ✓ Абстракции не должны зависеть от деталей реализации, напротив, детали реализации могут зависеть от абстракций.

- R.Martin, 1996

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 41



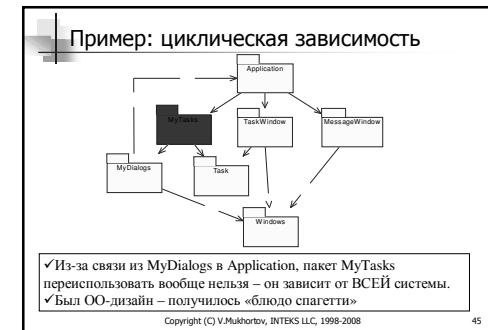
ADP – Ациклические связи

- ✓ Граф зависимостей между компонентами ПО (классы, пакеты, методы) должен быть ациклическим.

- R.Martin, 1996

- ❖ Две сущности которые не могут существовать друг без друга не могут быть (пере-)использованы иначе как вместе. В чем же тогда смысл в их разделении на различные классы (пакеты, методы)?
- ☐ Упрощает поддержку (maintainability)

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 44



Принципы дизайна пакетов

- > **CRP – Common Reuse Principle**
Общий принцип переиспользования
- > **CCP - Common Closure Principle**
Принцип локализации изменений
- > **SDP - Stable Dependencies Principle**
Принцип стабильности зависимостей
- > **SAP - Stable Abstractions Principle**
Принцип стабильности абстракций

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 46

CRP – Common Reuse Principle

- ✓ Классы из пакета должны переиспользоваться вместе. Пользователи должны зависеть от пакета в целом, а не от его части. - R.Martin, 1996
- ❖ ISP, адаптированный к пакетам
- ☐ Облегчает поддержку (maintenance)

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 47

Нарушение CRP: remote service

Проблема: Всякий раз когда выходит новая версия Service, клиенты ServiceAgent должны ожидать что их код может перестать работать, даже если изменения реально не затронули ServiceAgent.

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 48

CRP совместимое решение

Клиенты ServiceAgent зависят только от того что реально используют.
Выгоды: изменения в пакетах local и server не затрагивают клиентов

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 49

CCP: Локализация изменений

- ✓ Классы в пакете должны быть подвержены одному и тому же типу изменений – либо открыты для данного вида модификаций, либо закрыты от него. - R.Martin, 1996
- ☐ Локализует изменения и снижает число версий.

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 50

SDP – Стабильность зависимостей

- ✓ Пакет должен зависеть только от пакетов, более стабильных, чем он сам.
- ✓ нестабильность пакета – мера вероятности появления в нем изменений вследствие изменений других пакетов. - R.Martin, 1996

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 51

Нестабильность пакета

Нестабильность пакета:
 $I = Ce / (Ca + Ce)$

Где:
Ce = число исходящих связей (число классов вовне пакета, от которых зависят классы внутри пакета). = насколько пакет «зависим» от других пакетов
Ca = число входящих связей (число классов вовне пакета которые зависят от классов внутри пакета). = насколько пакет «важен» для других пакетов

I = 0 – абсолютно стабильный пакет
 I = 1 – очень нестабильный пакет

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 52

SAP – Стабильность абстракций

- ✓ Абстрактность пакета должна быть пропорциональна его стабильности. - R.Martin, 1996
- ❖ если все пакеты стабильны – система немодифицируема.
- ❖ => некоторые пакеты просто обязаны быть нестабильными.
- ❖ вопрос: какие это пакеты?
- ☐ Упрощает поддержку (maintainability)

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 53

Абстрактность пакета

Абстрактность пакета
 $A = Na / N$

где
Na = число абстрактных классов (интерфейсов)
N = полное число классов в пакете

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 54

Генеральная линия

- ✓ Строим график I(A) - нестабильность(абстрактность)
- ✓ Пакеты вдоль линии (0,1) to (1,0) имеют хороший баланс
- ✓ Отклонение от генеральной линии:

$$D = |A + I - 1|$$

указывает на потенциальные проблемы в дизайне пакета.

- R.Martin

Расчет I, A и D для Java реализован в утилите **Jdepend**

Copyright (C) V.Mukhortov, INTEKS LLC, 1998-2008 55

