# Аспект-ориентированное программирование

*к.т.н. Черкашин Е.А.,*

*лаб. 4.1. Комплексных информационных систем*

*ИДСТУ СО РАН*

# Фрагмент кода C# (1)



```
public class InvoiceService {

    . . . . .

    void CreateInvoice(ShoppingCart cart) {

        _transaction.Start();

        _invoicedb.CreateNewInvoice();

        foreach(item in cart)

            _invoicedb.AddItem(item);

        _invoicedb.ProcessSalesTax();

        _transaction.Commit();

        }
```

# Фрагмент кода C# (2)



```
_transaction.Start();
_invoicedb.CreateNewInvoice();
_logger.LogInfo("Started Sales Tax Processing");
foreach(item in cart)
    _invoicedb.AddItem(item);
    _invoicedb.ProcessSalesTax();
    _transaction.Commit();
_logger.LogInfo("Finised Sales Tax Processing");
```

# Фрагмент кода C# (3)



```
_userPermissions.CheckFor("tax-processing");
_transaction.Start();
_invoicedb.CreateNewInvoice();
_logger.LogInfo("Started Sales Tax Processing");
foreach(item in cart)
    _invoicedb.AddItem(item);
    _invoicedb.ProcessSalesTax();
    _transaction.Commit();
_logger.LogInfo("Finised Sales Tax Processing");
```

# Проблемы сквозного кода



- Программа сложна для понимания
  - Запутанность кода,
  - Рассредоточение кода,
  - Плохое прослеживание назначения;
- Непригодность для повторного использования;
- Большая вероятность ошибок;
- Трудоемкость сопровождения.

# История технологии



1. 1997 - Разработано в PARC, a Xerox Company;
2. 2001 - AOP в Java - AspectJ;
3. 2003 - **Распространение в другие системы программирования**.

**Другие технологии**

1. Dependency Inversion - обратные зависимости;
2. Python, D ... - декораторы;
3. LogTalk - перехват сообщений;
4. Model Driven Architecture - разработка ПО, управляемая моделированием.

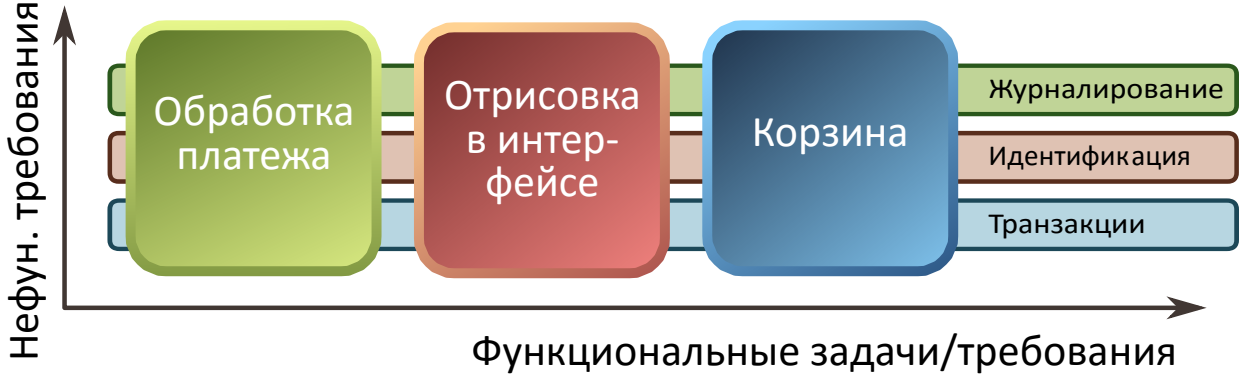
AOP - Aspect-oriented programming. PARC - Palo Alto Research Center WEB-site.

# Определения



**Функциональные требования** - требования, "имеющие ценность в проекте", т.е. функции разрабатываемой системы, решающие конкретную поставленную задачу обработки информации.

**Нефункциональные требования** не имеют прямого отношения к решаемой задаче, но должны быть выполнены, чтобы обеспечить корректное функционирование системы, создать среду для ее сопровождения.



# Примеры нефункциональных задач



- Синхронизация;
- Проверка ограничений;
- Отлов ошибок и их коррекция;
- Зависимости между функциями;
- Управление памятью;
- Проверка структур данных;
- Хранение объектов;
- Обработка транзакций;
- Перевод сообщений, локализация;
- Защита информации;
- Кэширование данных;
- Ведение журнала событий;
- Мониторинг событий/состояния;
- Правила предметной области (bussiness logics);
- Миграция кода;
- Оптимизация алгоритмов;
- Контрольные точки;
- Точки сохранения/восстановления состояния.

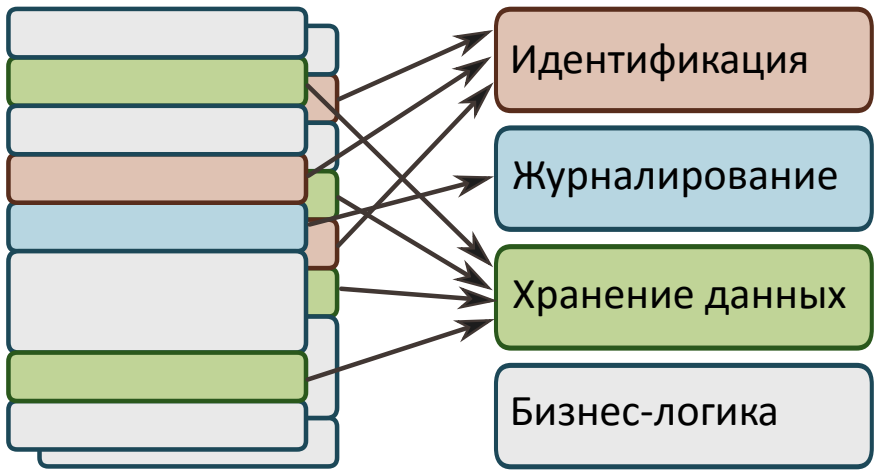
# Определения



**Сквозные задачи (cross-cutting concern)** - однотипные задачи, решаемые в разных частях программной системы.

**Спутывание (tangling, переплетение)** - "стиль" программирования, где несколько функций реализуются в одном коде одновременно.

**Разбрасывание (scattering)** - распределение решения задачи между отдельными процедурами (C-c,C-v).



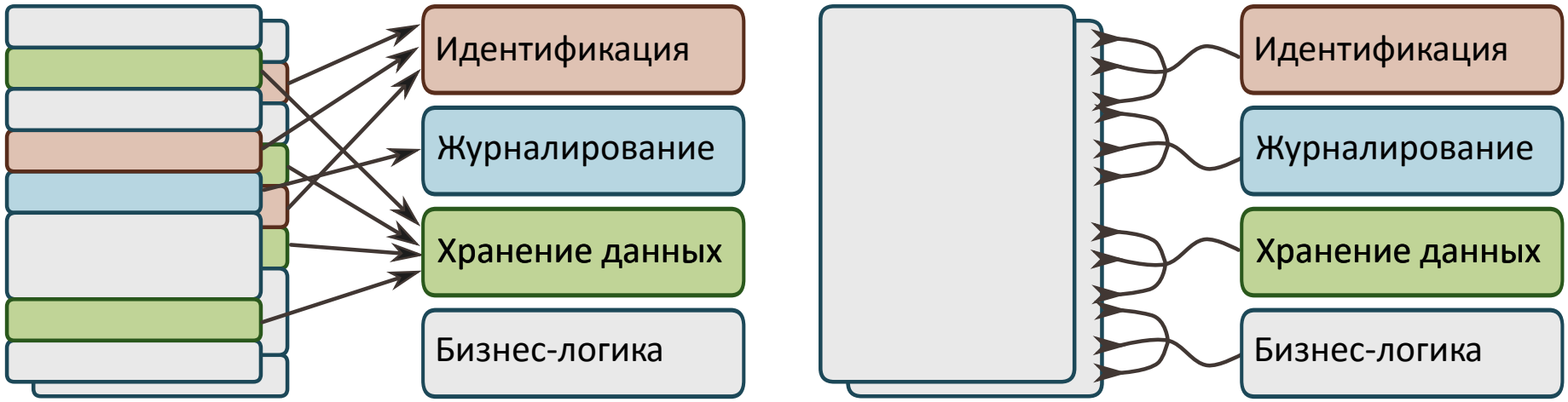
# Определения



**Сплетение (weawing)** - процесс порождения спутанного кода в соответствии с рекомендациями.

**Внедрение (introduction)** - Изменение структуры класса или иерархии наследования с целью реализации аспекта.

АОП

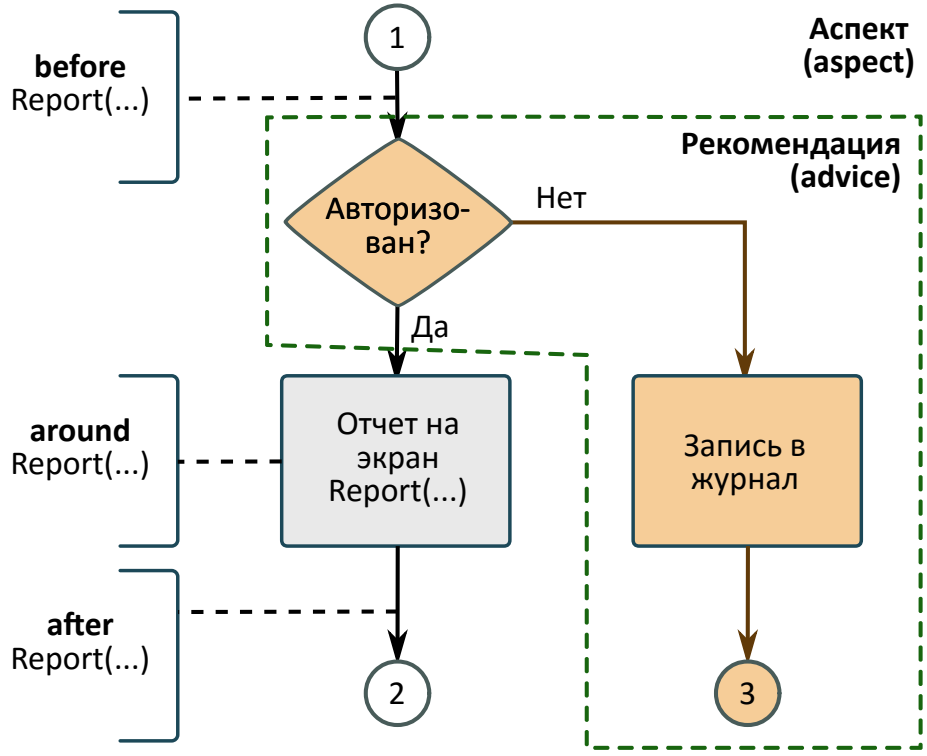


# Определения



**Точка соединения (join point)** - точка между операторами в коде программы или этапами исполнения некоторой процедуры.

**Срез (pointcut)** - множество точек соединения, определенное перечислением или ограничением; место, куда встраивается advice.

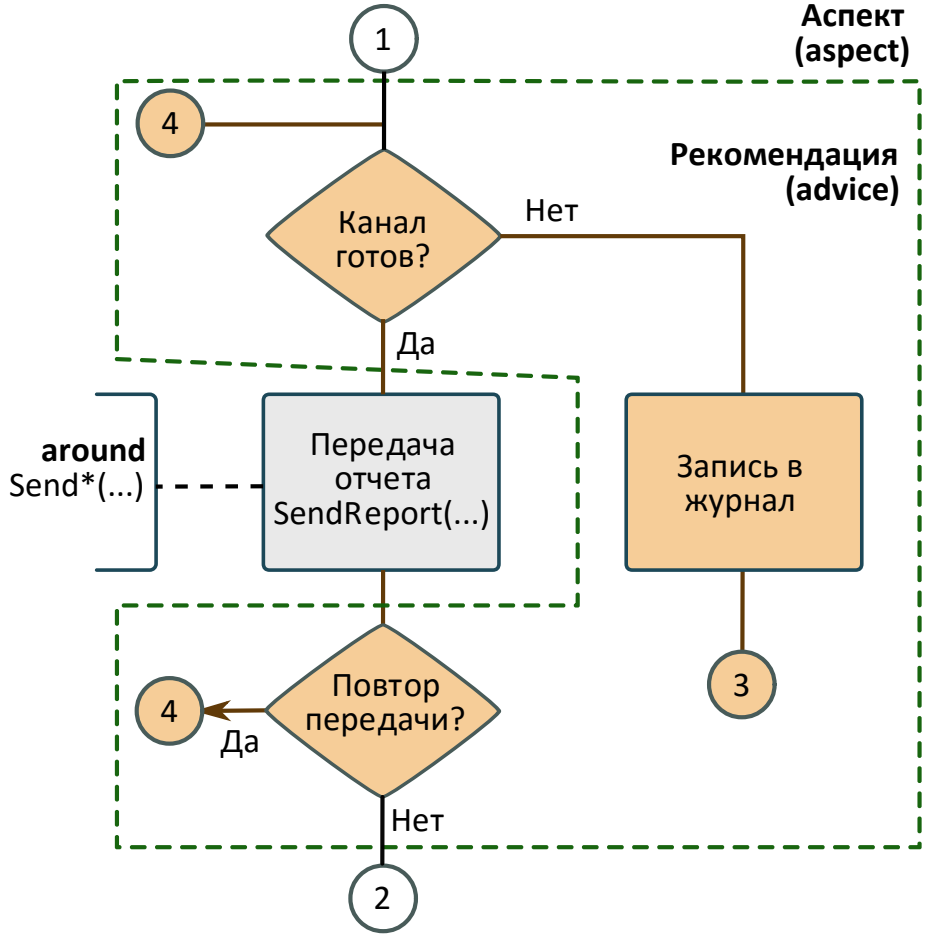


# Определения



**Рекомендация (advice)** - конкретный код, который выполняет заданную работу в заданном месте.

**Аспект (aspect)** - экземпляр программного кода для конкретной точки соединения (среза).



# Пример аспекта AspectJ - журналирование



```
public aspect AutoLog {
  pointcut pubMethods(): execution(public * org.cactus..*(..));
  pointcut logObjectCalls(): execution(* Logger.*(..));
  pointcut loggableCalls(): pubMethods() && !logObjectCalls()
  before(): loggableCalls() {
    Logger.entry(thisJoinPoint.getSignature().toString()); }
  after(): loggableCalls() {
    Logger.exit(thisJoinPoint.getSignature().toString()); }
}
```

# Точки соединения AspectJ (joint points)



- Вызовы методов;
- Обращение к членам класса;
- Блоки обработки исключений и др.;
- Другие точки соединения; например, результат выполнения метода передается другому методу.

```
void around(): call(public void Hello.say()) {
    if (Math.random() > .5) {
        proceed();      // запуск метода
    } else {
        System.out.println("В этот раз не повезло.");
    }}
```

# Инверсия зависимости (DI)



```
public class InvoiceService {

    . . . . . .

    InvoiceService(IInvoiceData invoicedb,
            ITransactionManagementService transaction) {
        _invoicedb = invoicedb; _transaction = transaction;}

    void CreateInvoice(ShoppingCart cart) {
        _transaction.Start();
        _invoicedb.CreateInvoice(cart);
        _transaction.Commit();
```

# Декораторы (Адаптер интерфейса)



```
public class TransactionDecorator: IInvoiceData {
    IInvoiceData _realService;
    ITransactionManagementService _transaction;
    public TransactionDecorator(IInvoiceData svc,
        ITransactionManagementService trans) {
            _realService = svc; _transaction = trans;}
    public void CreateInvoice(ShoppingCart cart) {
        _transaction.Start();
        _realService.CreateInvoice(cart);
```

# Декораторы Python (flask)



```
@app.route('/')
@templated('index.html')
def index():
    return dict(value=42)

@app.route('/user.html')
@templated()
@memcached(sec=20)
def index():
    return dict(value=42)
```

```
@app.route('/secret_page')
@templated('login.html')
@login_required
def secret_page():
    return dict(msg=
        'Please, login')
```

# Декораторы в Lisp



```
(defun attributes (keywords function)
  (loop for (key value) in keywords
        do (setf (get function key) value))
  function)
(attributes '((version-added "2.2")
    (author "Rainer Joswig"))

(defun add-func (a b)
  (+ a b))
)
```

# Перехват сообщений (LagTalk)



```
:- object(tracer,
    implements(monitoring)). % Протокол обработчиков сообщений.
    before(Object, Message, Sender) :-
        write('call: '), writeq(Object), write(' <-- '),
        writeq(Message), write(' from '), writeq(Sender), nl.
    after(Object, Message, Sender) :-
        write('exit: '), writeq(Object), write(' <-- '),
         writeq(Message), write(' from '), writeq(Sender), nl.
:- end_object.
```

# Перехват сообщений (LagTalk)

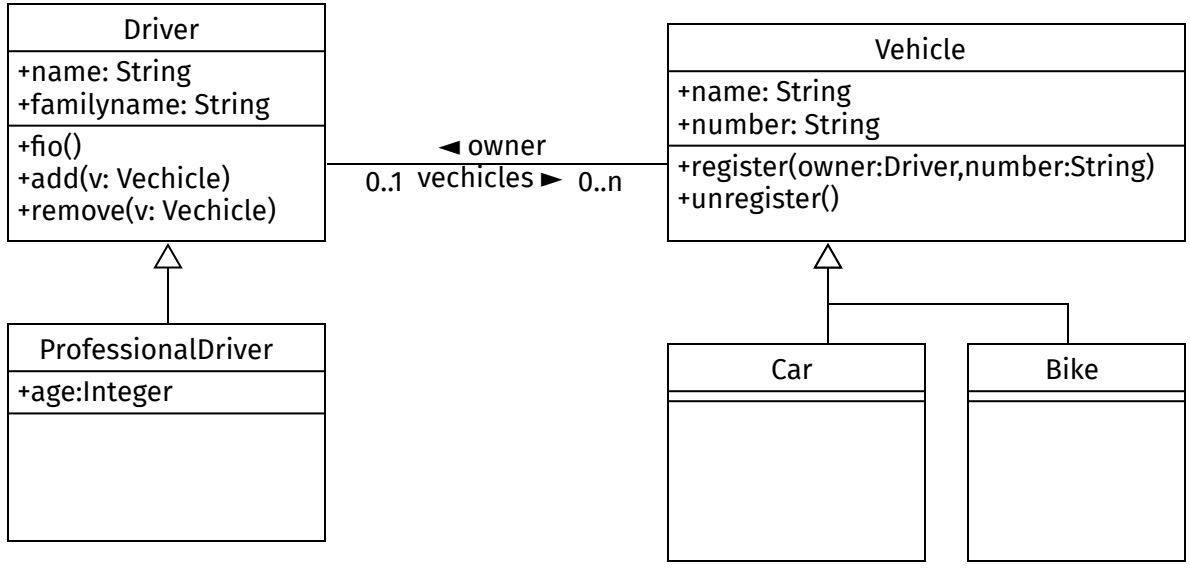


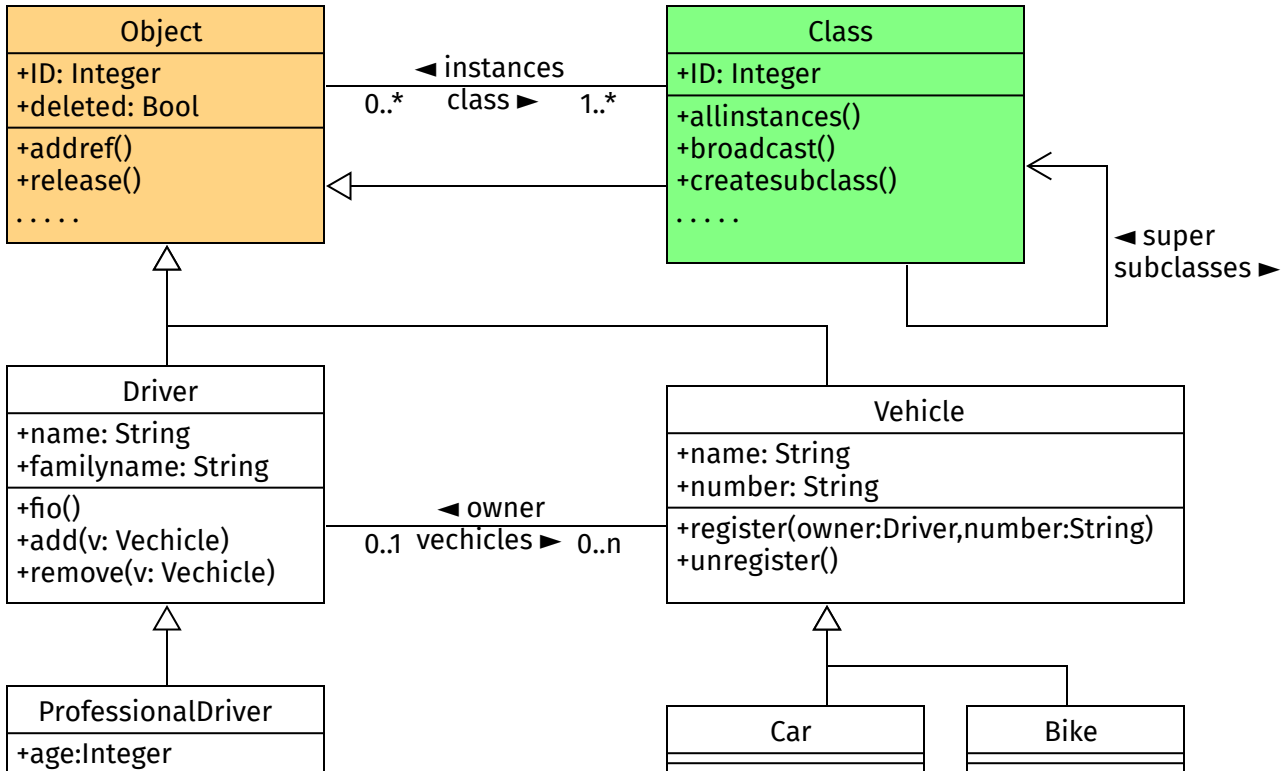
```
:- object(any).
    :- public(msg/1) .
    :- public(gsm/1) .
    msg(ok).
    gsm(no).
:- end_object.

| ?- define_events(_, _, _, _,
       tracer).
yes
```

```
| ?- any::bar(X).

call: any <-- msg(X)
 from user
exit: any <-- msg(ok)
 from user
X = ok
yes
```

# Model Driven Architecture






Object
+ID: Integer
+deleted: Bool
+addref()
+release()
. . . . .
◄ instances
0..*
class ►
1..*
Class
+ID: Integer
+allinstances()
+broadcast()
+createsubclass()
. . . . .
◄ super
subclasses ►
Driver
+name: String
+familyname: String
+fio()
+add(v: Vechicle)
+remove(v: Vechicle)
◄ owner
0..1
vechicles ►
0..n
Vehicle
+name: String
+number: String
+register(owner:Driver,number:String)
+unregister()
ProfessionalDriver
+age:Integer
Car
Bike

# Что посмотреть?



1. Функциональное программирование на Lisp/Scheme, ML, Haskell, Julia;
2. Объектно-ориентированное программирование (ООП), C++, Object Pascal, SmallTalk;
3. Язык программирования Python 2.7/3.6;
4. AOP для .NET, AspectJ и др. языков программирования;
5. Компонентные архитектуры и компонентное программирование, COM+, Mozilla XPCOM, Java Beans, Delphi (Lazarus), Zope component architecture;
6. Пролог и LogTalk;
7. Стоит ознакомиться с D, Go, Rust.